

オープンデザイン

OPEN DESIGN

No. 1

集中
特集

SCSI完璧リファレンス



緊急企画●PCMCIA(PCカード)詳細解説

オープンデザイン No.14 (’96年6月号) 好評発売中

集中特集 最新の暗号技術によるセキュリティの実現



B5判 152頁 定価1,800円
貴重なデータがネットワークを流れるようになり、クラッカーが暗躍し始めました。いわゆる現代は、情報戦争状態にあるということを認識したうえで、EC(Electronic Commerce)やEDI(Electronic Data Interchange)の実現には、ファイア・ウォールに代表される守りの安全性だけでなく、もっと攻撃的な安全性の追求が必要です。この特集では、その要となる最新の暗号化技術によるセキュリティの実際を研究します。

- ワンタイム・パスワード・システム
- MOSS
- Netscape関連の暗号化技術
- SSL, S/MIME, Secure IP
- 暗号化技術全般に関する概説

既刊好評発売中

No.1 集中特集 SCSI完璧リファレンス

PCやWSをはじめ、スーパーコンピュータにまで装備されている外部機器接続のための標準規格：SCSIについて整理。

No.2 総力特集 Macintoshインターフェース&ネットワークング

Macintoshのインターフェースとネットワークング技術を集集。

No.3 集中特集 イーサネットとTCP/IP

イーサネット・カード、ドライバ、TCP/IPプロトコル・スタック、TCP/IPとアプリケーションについて整理。

No.4 全面特集 UNIXネットワークング実践編—インターネットのために

「LANの種類と接続」や「モデムを介した遠隔操作」について整理。

No.5 総力特集 ネットワーク・ファイルシステムの研究—NFS/NIS/RPC

ネットワーク・ファイル・システムの構造を調べ、NFSの利用法やPC-NFSについて整理。

No.6 集中特集 インターネットワークのトラブル対策

ネットワーク・アナライザの種類と活用法、物理層/データリンク層、TCP/IPのあらましとトラブル対策です。

No.7 全力特集 PCIバスの詳細と応用へのステップ

PCIの概要からアービトレーションやインタラプト、コンフィギュレーションや64ビット拡張について解説。

No.8 全面特集 電子メール・システム完全マスタ

インターネットの普及によって注目されている電子メールの仕組みや実現法を整理。

No.9 連続特集 電子メールの導入・相互接続とグループウェア

パソコン電子メールの導入事例や、インターネット電子メール・システムとの相互接続について解説。

No.10 全力特集 ネットワーク管理技術のすべて

オープンシステムのネットワーク管理を正しく、効率的に行うための技術を集集。

No.11 集中特集 クライアント/サーバ・システムの実現—RPCとIPC

クライアント/サーバ・モデルの実際について、とくにRPCとIPCを中心に解説。

No.12 総括特集 LAN技術総合入門

もう一度原点に立ち戻り、ネットワークングの基本ともいえるLANの技術を確認するための特集。

No.13 集中特集 HTMLリファレンス

HTMLという記述言語を使って個性的なホームページを作成するためのコマンドをまとめた。



集中特集 SCSI完璧リファレンス

最近、パソコン・システム自体がマルチベンダで構成されるようになってきており、アダプタ・カードや周辺機器とのコネクティビティが問題になるなど、パソコン自体のオープン性が大きなテーマとなりつつあります。そして、今、パソコンのコネクティビティ技術でもっとも注目されているのが、SCSI…スカジーといわれる標準パラレル・インターフェースです。SCSI規格で接続できない周辺機器が見あたらないほどに一般的なインターフェースになりました。

SCSIポートをもつホストもパソコンやワークステーションだけに限りません。SCSI-1から、SCSI-2、-3と規格が改訂され、高速化/高性能化がはかられるにつれ、スーパーコンピュータにもSCSIをもつマシンが登場しており、いまやSCSIはコンピュータのサイズに関係なく、外部機器接続のための標準規格になりました。

OPENDESIGN創刊号では、このSCSIを大々的にとりあげます。標準規格だから、コネクタをつなぎさえすれば動く、とはいかないのが異機種間接続の厄介なところで、SCSIもその例にもれません。そこで、SCSIで外部機器をつなぐ機会の多いオープンシステムのエンジニアのために、

- ▶ SCSI装置の動作とコマンドに関する知識
 - ▶ パソコンやワークステーションの機種ごとのSCSIの特徴やSCSIデバイス・ドライバの作り方
 - ▶ 遭遇しやすいトラブル事例とその対策
- などをポイントに、SCSIの技術とノウハウを特集します。

緊急企画 PCMCIA(PCカード)詳細解説

SCSIカード、イーサネット・カード、モデム・カードなど、ノートパソコンの外部機器接続で急浮上しつつあるのがPCMCIA、別名PCカードです。SCSI関連としてここでは、PCMCIAの位置づけから、ハード仕様/カード・サービス/ソケット・サービスそして応用例までをとりあげます。

集中特集 SCSI 完璧リファレンス

第1章 SCSI はじめの一步 大島啓孝4 バス構成/信号/バス・フェーズなどを簡潔に記述

- ① SCSI とは?
- ② SCSI バス・フェーズ

第2章 SCSI をもう一步踏み込んで理解する 大島啓孝20 データ転送/バス条件/SCSI ポインタ/メッセージ・システム/ステータス/共通コマンド詳説

- ① データ転送 (FAST SCSI, WIDE SCSI)
- ② SCSI バス・コンディション (バス条件)
- ③ SCSI ポインタ
- ④ メッセージ・システムの詳細
- ⑤ ステータス
- ⑥ コマンド・オーバビュー

第3章 主な SCSI 装置の動作とコマンド 二上貴夫/大島啓孝44 SCSI デバイスの動作やデータの記録/再生のメカニズムをコマンドとからめて理解する

- 3-1 ダイレクト・アクセス・デバイス 45
- 3-2 CD-ROM デバイス 48
- 3-3 スキャナ・デバイス 64
- 3-4 QIC ストリーマ 75

第4章 機種別 SCSI 研究80 98・AT・UNIX/他機種用がなぜつながらない/互換性の阻害要因は?/ デバイス・ドライバはどうつくる

- 4-1 98 の SCSI と 55 ボード&92 ボード 真樹美智 80
- 4-2 AT 互換機と ASPI プログラミング 鈴木祥夫/吉野 誠 91
- 4-3 WS/UNIX 上で SCSI ドライバを作成する 高須俊介 103



第5章 98&AT 用 SCSI ユーティリティ 真樹美智115

互換性チェックや解析に威力を発揮するデバッグ・ツールの機能と使い方

- ① 開発の動機
- ② SU.EXE の機能概要
- ③ SU.EXE の使用方法
- ④ SU.EXE のコマンド詳細

Appendix 汎用計測器と併用する SCSI 簡易ツールの製作 有吉和久 138

第6章 SCSI トラブル相談室 有吉和久／清水哲夫／真樹美智／吉田浩幸141

失敗例に学ぶ SCSI システム活用/構築ノウハウ

SCSI-1 と SCSI-2 の混在使用の注意点は？/FAST SCSI で複数デバイスをつなぐとエラーが起きやすいが？/98 の 55 ボードで HDD や MO 以外の機器がつながるか？/98 の 92 ボードにサード・パーティ製ディスクを接続したらハングアップしてしまうが？/ほか

Appendix エラー・コード(ASC/ASCQ)一覧 156

第7章 規格化進む SCSI-3 の概要 菅谷誠一158

P ケーブル/Q ケーブル/16 ビット幅転送/32 ビット幅転送/SCSI ID 拡張
コマンド拡張/メッセージ拡張

- ① インターフェース規定の概要
- ② 物理規定
- ③ プロトコル規定
- ④ メッセージ・システムの拡張

緊急企画

プラグ&プレイで急浮上の新規格

PCMCIA(PC カード)詳細解説 岡村周善167

PCMCIA とは/スロット/CIS とタブ/イネーブラとソケット・サービスとカード・サービス/PC カードの現状・問題点と今後の方向

SCSIはじめの一步

バス構成/信号/バス・フェーズなどを簡潔に記述

大島 啓孝

SCSIは、8ビットの第1世代SCSI-1、高速性/汎用性を向上させたSCSI-2となり、すでにSCSI-3も討議されつつある。SCSIでは、装置はデバイスとよばれ、命令を出す装置=イニシエータと、命令を受け取り実行する装置=ターゲットに分類されるが、これは固定的なものではない。SCSIバスの使用状態を示すバス・フェーズは、バス・フリー→アービトレーション→セレクションときて、コマンド/データ/ステータスなどを転送するフェーズへ…といった順序で変化する。各フェーズは、制御ラインの各信号の組み合わせで決まる。

(編集部)

1

SCSIとは?

■ SCSIのおいたち

SCSIとは、Small Computer System Interfaceの略で、スカジーと呼ばれています。SCSIは、パソコン用のフロッピーやハード・ディスク装置用に米国のシュガート社が作り出したSASI(Shugart Associates System Interface)に端を発し、関係各社によって汎用性が高められてANSI(American National Standard Institute)の規格、ANSI X 3.131-1986として完成されたかたちになりました。このインターフェースはその名のとおり小型コンピュータ・システム用、しかも補助記憶装置用に考えられたものですが、その汎用性の高さと使い勝手の良さ、そしてそこそこの性能をもっていたため、また各半導体メーカーがSCSIコントローラLSIをつぎつぎと市場投入したために、急速に普及しました。パソコン、ワークステーション、オフコン、ミニコン、さらにはスーパーコンピュータにいたるま

でのシステムで、標準I/Oインターフェースとしてその地位を築いてしまいました。

こうなると、さらなる性能向上と汎用性が望まれ(市場ニーズ、ハードウェアと規格のイタチごっこ)、現在では、ANSI X 3.131-1986がSCSI-1と呼ばれるのに対し、SCSI-2という拡張版が標準となりつつあります。SCSI-2は、SCSI-1に「高性能化」と「さらなる汎用性」を加え、そしてSCSI-1のもっていた「曖昧さ」を明確にしたものです。したがって、SCSI-1とSCSI-2はまったく違ったものではなく、とくに区別する必要がなければ一般的に「SCSI」と呼んでさしつかえありません。「SCSI-2規格はSCSI-1規格を包含している」ともいえます。本誌はSCSI-2をベースにできていますが、必要に応じてSCSI-1とSCSI-2の違いも取り入れて説明します。

なお、すでにSCSI-3という規格も検討されています。

■ SCSIの基本は「物理的仕様」と「論理仕様」

SCSI規格は、ケーブルやコネクタ、電気的信号などのあるべき姿を定めた「物理的仕様」とそのインターフェース上での機器間のやりとりの約束事(プロトコル)を定めた「論理仕様」よりなりたっています。基本はこの二つしかありません。基本の範囲が非常に広いので、逆にいえば汎用性・拡張性が大きいのです。この章は入門編なので、規格の細かいことにはこだわらずに話をすすめますが、基本である「物理的仕様」と「論理仕様」という概念がわかっていれば、規格書を見るときに役立つと思います。

*以後、とくに断らないかぎりSCSIはSCSI-2をさす。

■ SCSI に接続可能な装置をデバイスという

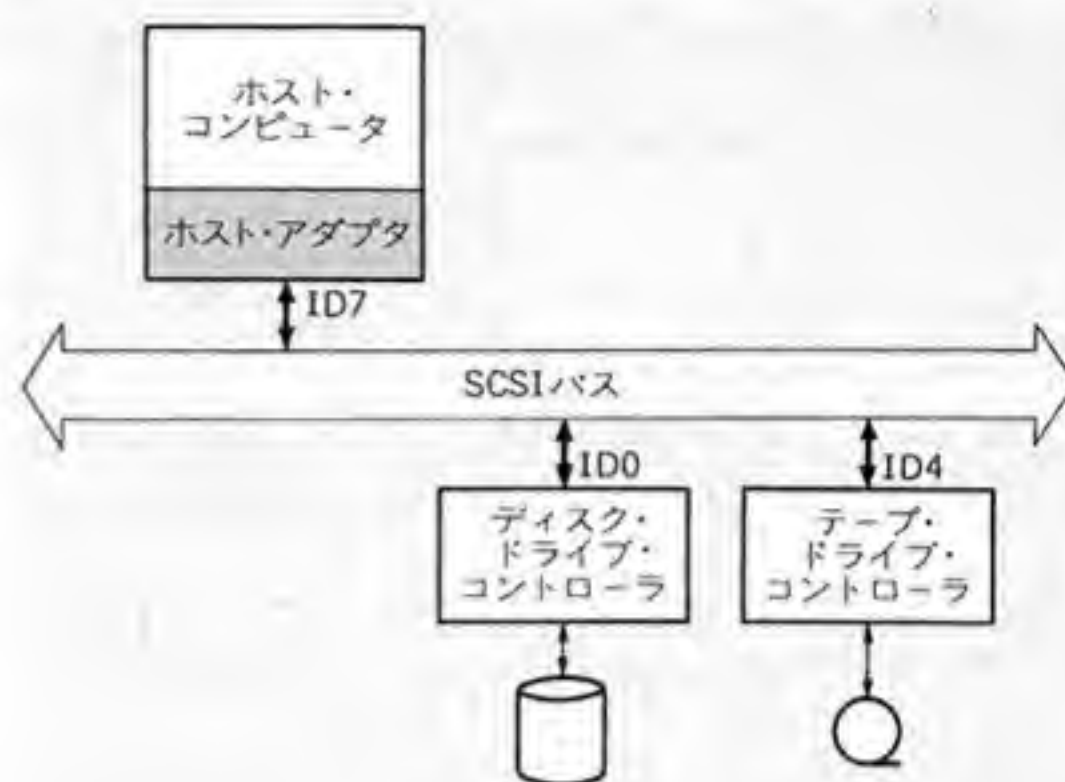
SCSI バスに接続された装置を「SCSI 装置(デバイス)」といいます。規格上定義されているデバイス(装置)は表1のようになっていますが、これはあくまで定義されているだけであって実際に使われているのは、ハード・ディスク、磁気テープ装置、MO(光磁気装置)、CD-ROM、プリンタなどの実例が多いようです。もちろん、ホスト・コンピュータも SCSI デバイスの一つです。しかし、表1からもわかるように、まだまだ拡張性があり、SCSI という共通の切り口をもっていれば、どんな機種も接続が可能です。SCSI で動く自動車や洗濯機ができて不思議ではありません(?)。

■ システム構成：SCSI バスに接続できるデバイスは全部で 8 台

1 本の SCSI バスに接続できる SCSI 装置は全部で 8 台で、それぞれの装置に ID(Identifier)といわれる 0~7 の認識番号をもっています。ここでいう「SCSI 装置」はディスク・ドライブとかホスト・コンピュータといった具体的な装置ではなく、SCSI バスに直接接続されているものと認識してください。

具体的な例としては、図1に示すようにホスト・アダプタを含んだホスト・コンピュータや SCSI 用のディスク・ドライブ・コントローラ、テープ・ドライブ・コントローラなどが SCSI 装置にあたります。こういった装置が全部で 8 台接続できるわけですから、ホスト・コンピュータが 1 台に対してディスク・コントロ

図1 システム構成例① ホスト 1 台と周辺機器 2 台の例



ーラが 7 台、さらに理論的には、ロジカル・ユニットと呼ばれる装置が 8 台まで接続可能です。おのこのディスク・コントローラにディスク・ドライブが 8 台ずつ(計 56 台のディスク・ドライブ)接続されたシステム(図2)も構成できますし、ホスト・コンピュータが 7 台あり、ディスク・コントローラ 1 台、ディスク・ドライブ 1 台(図3)のシステムも構成できます。もちろん、ホスト・コンピュータにターミナルが何台接続されていてもかまいません。

最近では、SCSI をもったディスク・ドライブやテープ装置が普及していますので、図4のような構成が現実的であるともいえますが、ここでは SCSI の基礎・基本を説明するため、あえて

表1 SCSI 規格で定義されているデバイスとコード

デバイス・タイプ・コード (Inquiry データに表示されるべきコード)	デバイス・タイプ
00h	ダイレクト・アクセス・デバイス (例: 磁気ディスク)
01h	シーケンシャル・アクセス・デバイス (例: 磁気テープ)
02h	プリンタ・デバイス
03h	プロセッサ・デバイス
04h	1 回書き・複数回読み出し・デバイス (例: ある種の光ディスク) (Write-Once, Read-Multiple Device=WORM デバイス)
05h	CD-ROM デバイス
06h	スキャナ・デバイス
07h	光メモリ・デバイス (例: ある種の光ディスク)
08h	メディア・チェンジャ・デバイス (例: ジュークボックス)
09h	通信デバイス
1Fh	未定義, またはデバイス・タイプなし

図2 システム構成例② 各デバイスに子装置を8台まで接続したシステム

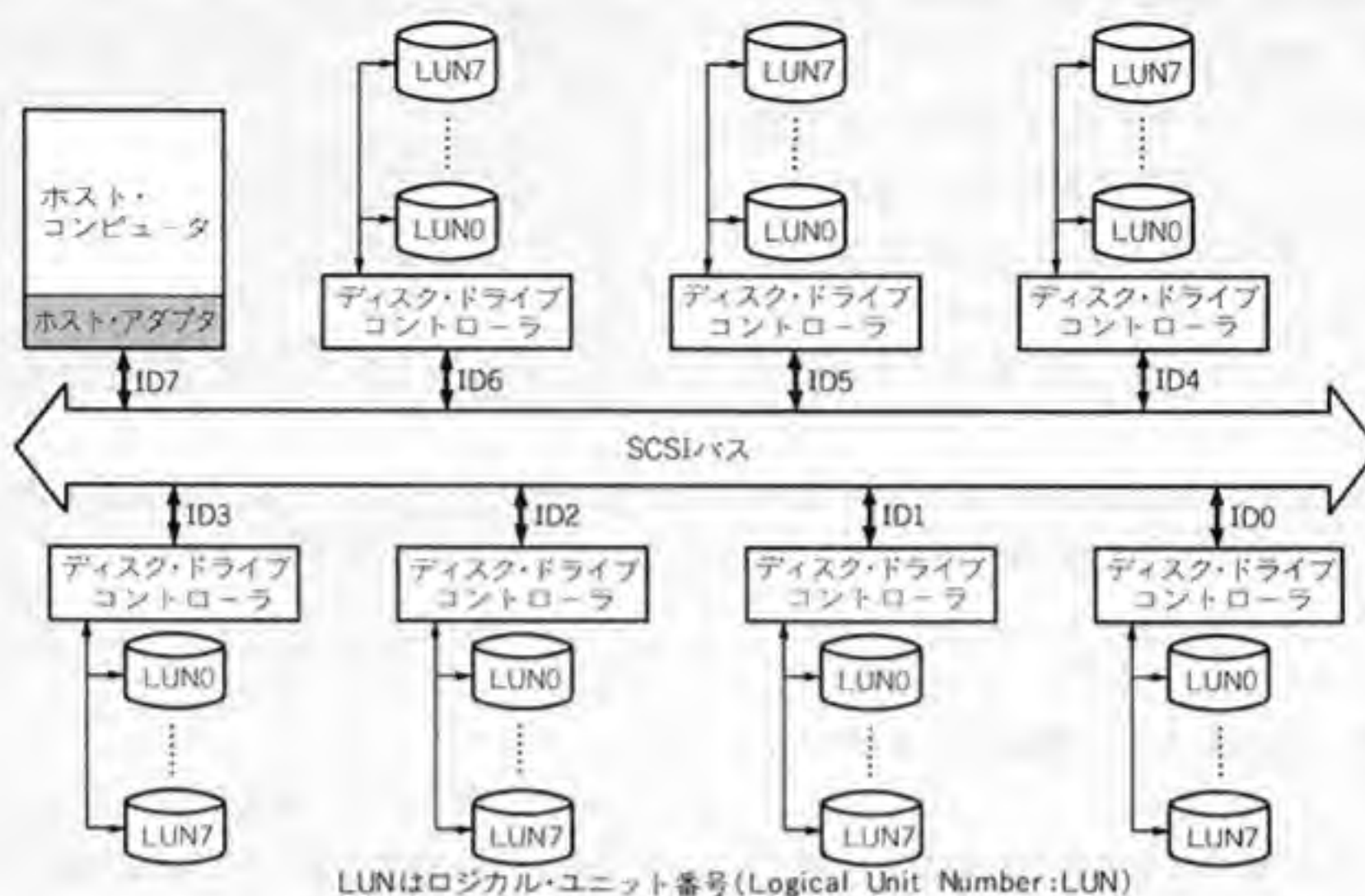
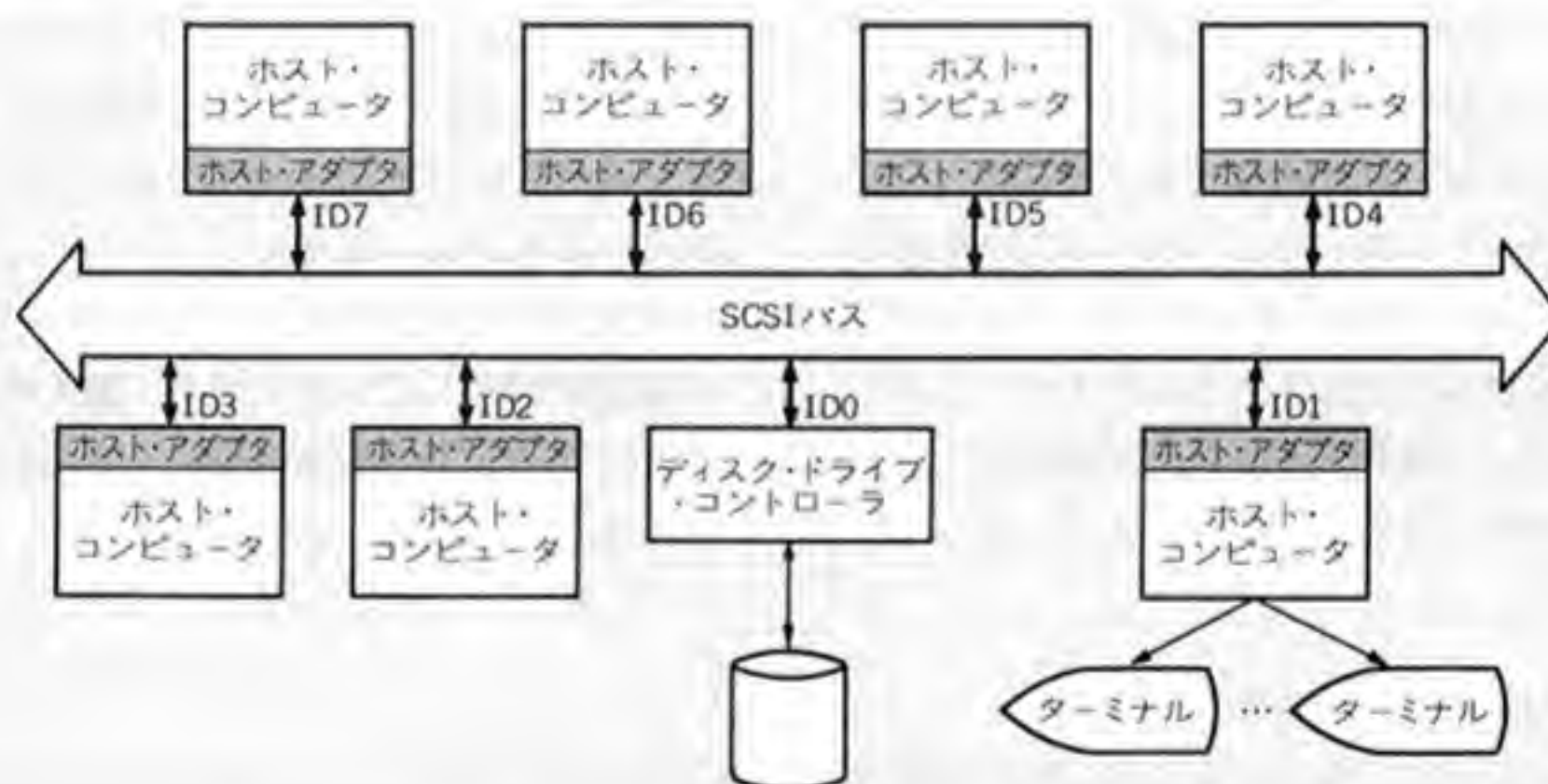


図3 システム構成例③ ホストが7台にデバイスが1台の例



SCSI バス→コントローラ→各種(ディスク、テープ
など)装置

という構成にしてあります。

さて、1本の SCSI バス上で合計8台までの SCSI 装置が稼働できるわけですが、実際動作する場合には、イニシエータとターゲットという、それぞれの装置の「立場」ができます。イニシエータとは命令を出す装置で、ターゲットはその命令を受け取り、実行する装置(実際に命令を実行するのは、ロジカル・ユニットといわれる子装置)です。

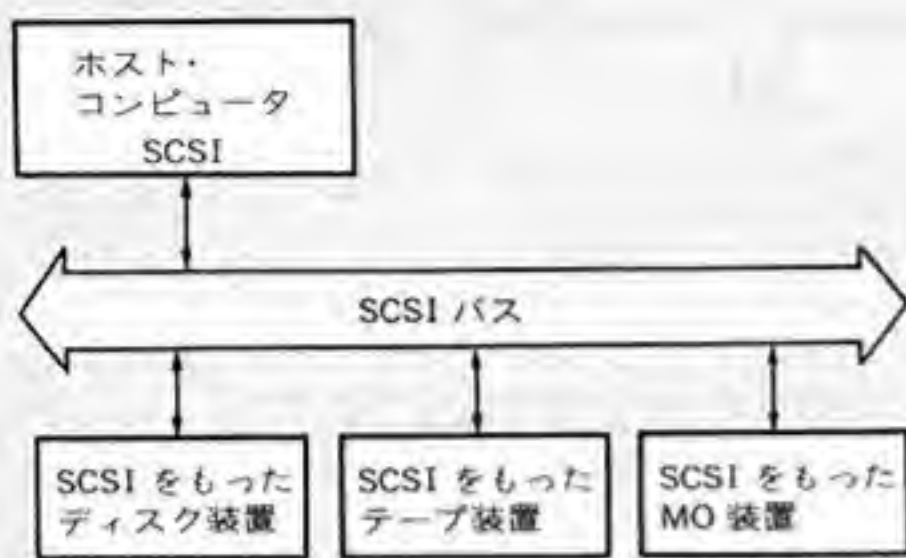
一般的な例では、図5のようにホスト・コンピュー

タがイニシエータで、ディスク・コントローラがターゲットになり、ホストからの読み出し命令にしたがってディスク・コントローラはディスク・ドライブからデータの読み出し(Read)を行います。

それでは、つねにホスト・コンピュータ=イニシエータ、ディスク・コントローラ=ターゲットでしょうか？

違います。SCSI バス上では、それぞれの装置がイニシエータにもターゲットにもなることができます。たとえば、図2の例でID0のディスク・コントローラがイニシエータとなり、ID2のディスク・コントローラ

図4 SCSI 内蔵装置でのシステム例

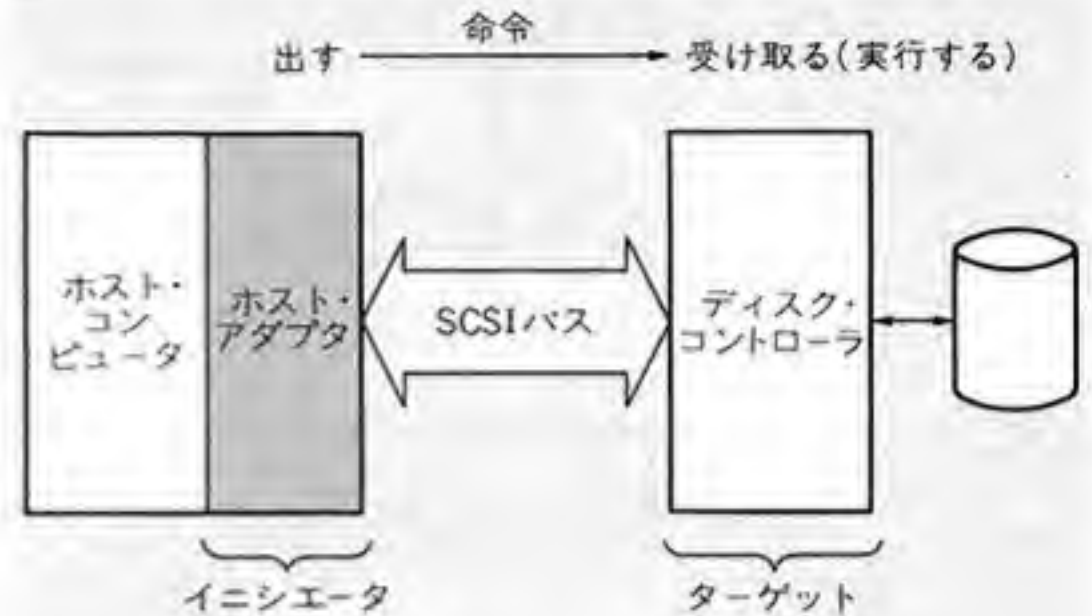


に接続されているディスク・ドライブに書き込むこともできます。つまり、「イニシエータ」、「ターゲット」というのは、「そのときどのように動いているか」で決まるもので「装置ごとにきめられてはいない」のです。しかしながら「どの装置も、イニシエータにもターゲットにもなることができる」というのは、SCSI 規格上の約束事であって、実際の装置では設計する際にターゲットとしての機能しかもたされなかったりすることが多いようです。

さて、システム構成例②(図2)にあるように、ターゲットとなるそれぞれの装置は、ロジカル・ユニットをもち、ロジカル・ユニットにはLUN(Logical Unit Number)が付けられます。したがって、イニシエータの命令は一つのターゲットに対し「どのロジカル・ユニットにデータを書き込め」とか「どのロジカル・ユニットから読み出せ」というようにLUNの指定が絶対に必要なわけです。

前の例はロジカル・ユニットが複数の場合でしたが、図4に示した例でもロジカル・ユニットは必ず一つ存在しなくてはなりません(これはSCSIの規約上の決まりなのです)。したがって、最近ではSCSI インターフェース*をもったハード・ディスクやMO、テープ装置などが非常に一般的ですが、この場合、LUNは一つ(“0”のみ)しか存在せず、したがって、たとえばSCSIをもったホスト1台に最大7台の周辺装置しかつなげないわけです(多少飛躍しますが、これは、SCSI コントローラ・チップが多くの中規模メーカーより出され、安く入手しやすくなったこと、また、ロジカル・ユニットの概念を軽視したことの弊害ともいえる)。

図5 イニシエータとターゲットの例



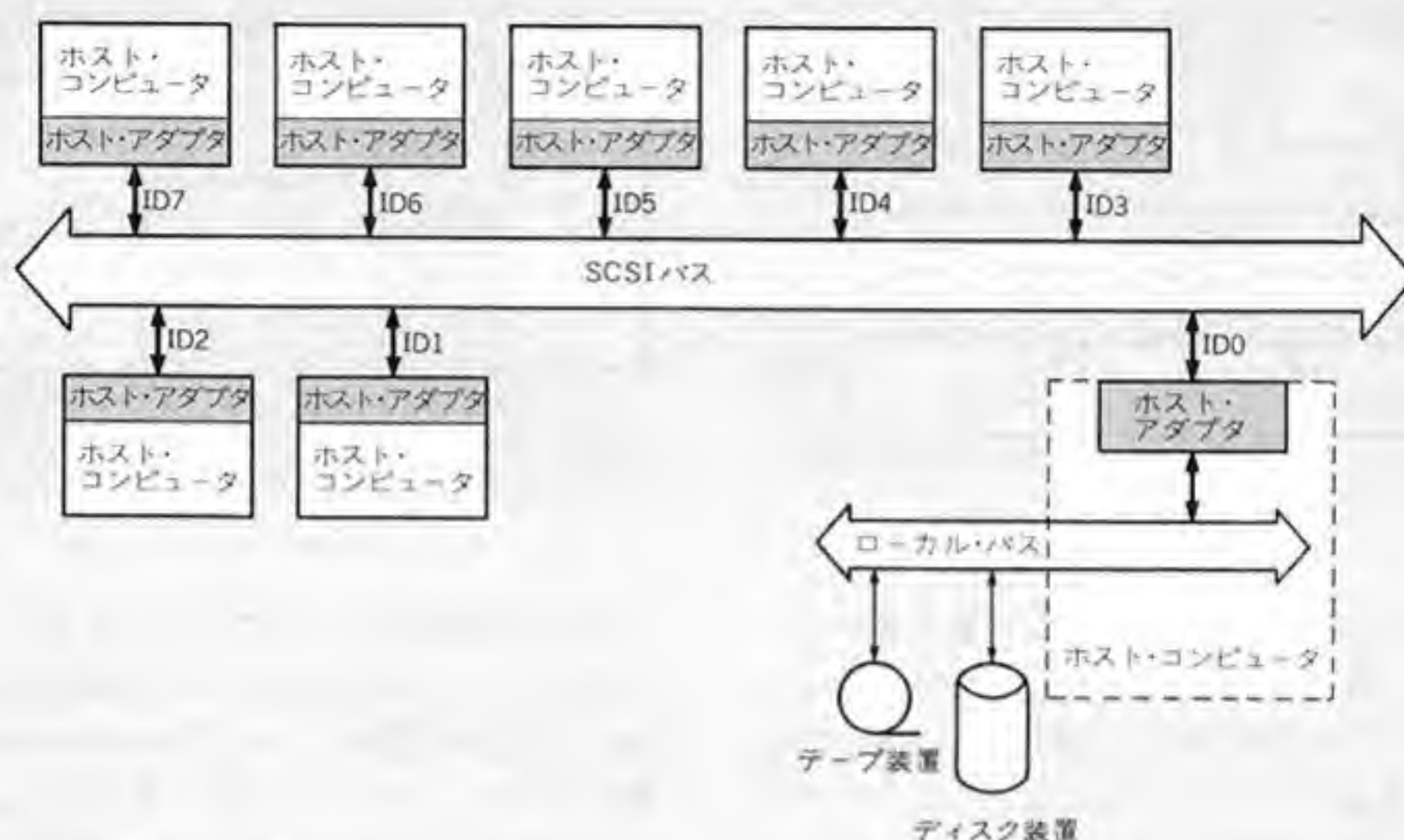
SCSI-2 規約では、一つのターゲットは、ロジカル・ユニットを八つまでもてますが、SCSI-1 の拡張仕様では、LUN が拡張され、一つのターゲットが最大 2048 個のロジカル・ユニットを持てるようにとまで考えられていました。SCSI-2 では、これを現実にとったかたちにするため、ロジカル・ユニットは最大 8 個までとあらためられました。SCSI-3 では、ロジカル・ユニットではなく、SCSI ID を 0~31 まで、すなわち 1 本の SCSI バスに最大 32 個の SCSI 装置が接続できるように考えられています。

これまでの説明で、SCSI を用いた場合、どのようなシステムを構成できるかがおおざっぱにわかったと思いますが、ひとつ極端な例を図6に示しましょう。まるでローカルエリア・ネットワークのような構成です。1本のSCSIバスに8台のホスト・コンピュータが接続され、その中の1台(ID0)だけがコンピュータ内のローカル・バスにディスク・ドライブとテープ・ドライブをもっています。このシステムでID0のコンピュータがターゲットとしての機能をもっていれば、どのホスト・コンピュータからでもこのID0に接続されているディスク・ドライブやテープ・ドライブを利用することができます。

このようにSCSIバスは、どの装置も対等の立場に立つことができる双方向バスで、非常に民主的なシステム・バスです。規格書に描かれているシステム構成の絵にとらわれず、SCSIのフレキシビリティをここで認識しておいてください。

* SCSIの“I”は「インターフェース」なのでSCSIインターフェースとはおかしい言葉ですが、このほうがわかりやすいのでヘンを承知で使います。

図6 システム構成例④ まるでLANのようなSCSI接続例



■ SCSI のバス構成、各信号の意味

ハードウェアの構成や物理仕様の細かいところは規格書にまかせるとして、ここでは各信号の意味・使い方を中心に説明しましょう。

SCSI のバス構成は、SCSI-1 時代からの 1 バイト (8 ビット) 転送を行う A ケーブル (50 ピン) と、SCSI-2 になって規格化されたオプションである 2 バイトまたは 4 バイト幅のワイド・データ転送を行うための B ケーブル (68 ピン) がありますが、B ケーブルの実用例がありません。SCSI の動作は A ケーブルのみによる (B ケーブルにはデータ・バス拡張の働きしかない) ので、以下 A ケーブル上の信号のみをとりあげます。

SCSI の A ケーブルには 8 本のデータ・ラインと、1 本のデータ用パリティ・ライン、および 9 本の制御用ラインがあります (図 7)。

この 9 本の制御用ラインの信号は以下のように大別できます。

- データ転送のタイミングを制御する信号…REQ (Request), ACK (Acknowledge)
- データ・バスの使い方を決める信号…MSG (Message), SEL (Select), C/D (Control/Data), I/O (Input/Output)
- その他の信号…BSY (Busy), ATN (Attention), RST (Reset)

では、これらの信号の機能について説明しましょう。

(1) REQ, ACK 信号とデータ・ライン

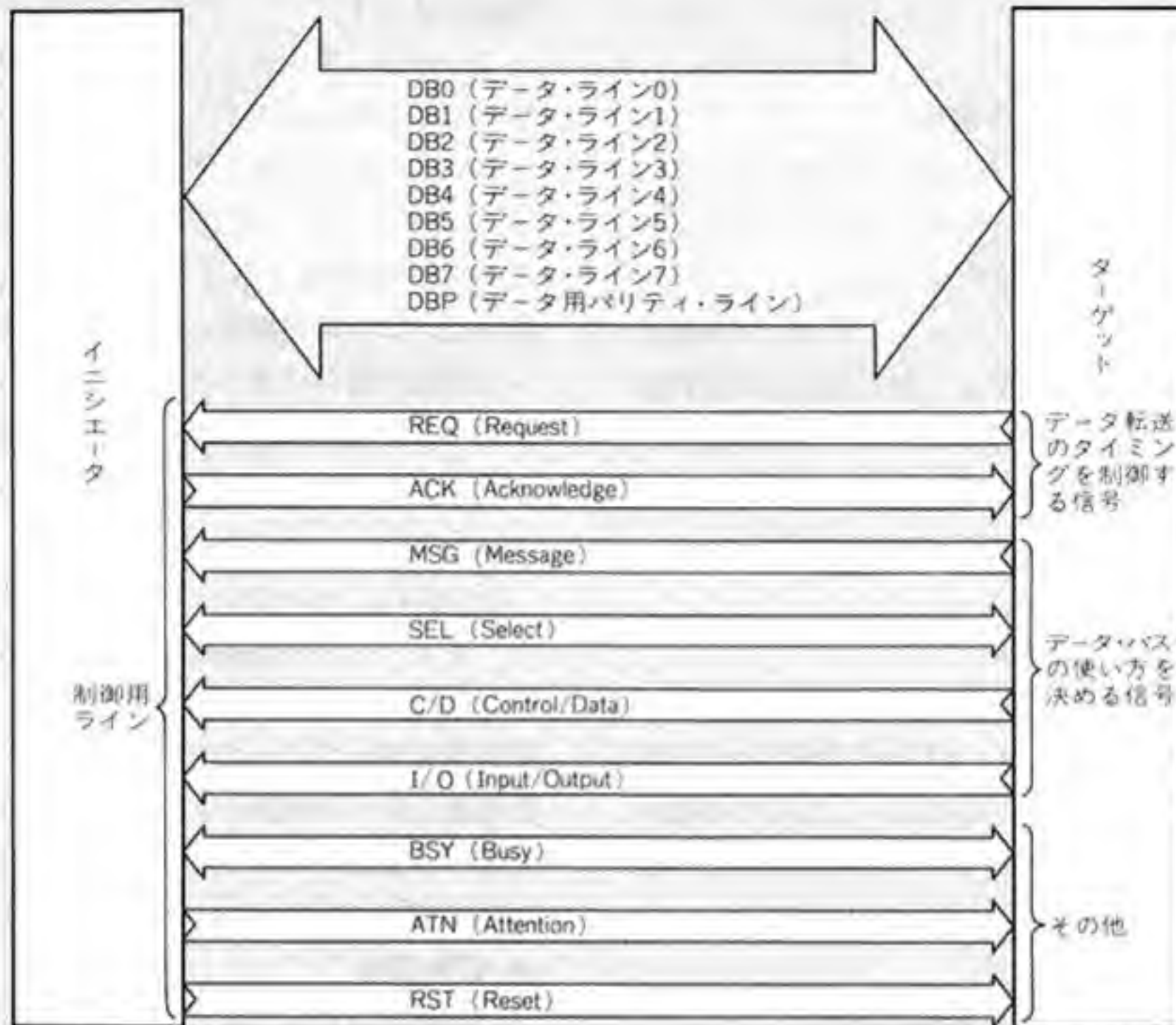
DB0~DB7 はデータ・ラインで、このラインに 8 ビットのデータがのり、このデータはターゲットが出力する REQ とイニシエータが出力する ACK によるハンドシェイクで転送されます。

具体的には以下のような順序になります。これは、非同期転送の例ですが、基本なのでしっかり理解しておきます。

シーケンス A イニシエータ→ターゲットの 1 バイト・データ転送 (Data Out など)

- ① ターゲットが REQ を “真” にする。
↓
- ② イニシエータは出力したいデータを DB0~DB7 にセットする。
↓
- ③ イニシエータが ACK を “真” にする。
↓
- ④ ターゲットは、ACK が “真” になるのを見てデータを取り込み、REQ を “偽” にする。
↓
- ⑤ イニシエータは REQ が “偽” になったのを見て ACK を “偽” にし、つぎの REQ を待つ。

図7 SCSIの信号



注) SCSIの信号方式は①シングルエンド(不平衡)型と②ディファレンシャル(平衡または差動)型の2種類がある。不平衡型はオープン・コレクタ出力、アクティブLow("L"="真")である。一方、平衡型(差動型)は、一つの信号に対し+信号と-信号の2本が使われ、+信号のレベルが-信号のレベルより高くなった場合が"真"と認識される。現在のところ、不平衡型のほうが平衡型よりも一般的である。そこで本章の説明では、不平衡型の信号とし、"L"="真"、"H"="偽"とする。

(複数バイト転送する場合、この①~⑤をくり返す。)

シーケンスB ターゲット→イニシエータの1バイト・データ転送(Data In など)

①ターゲットは出力したいデータをDB0~DB7にセットする。

↓

②ターゲットはREQを"真"にする。

↓

③イニシエータはREQが"真"になったのを見てデータを取り込み、ACKを"真"にする。

↓

④ターゲットはACKが"真"になったのを見て

REQを"偽"にする。

(つぎのデータがある場合、ACKが"偽"になるのを待たずに①へ戻ることができる。ただし、ACKが"偽"になっていないと②のREQは"真"にできない。)

上のA、Bのシーケンスを見て「おや?」と思われませんか? そうです。データ転送中の主導権はターゲットにあるのです。つまり、イニシエータはターゲットの要求がなければデータを送れませんし、また、イニシエータからデータを要求することはできず、ターゲットから送られるまま受け取らなければなりません。

これは SCSI の特徴のひとつで、SCSI バスの動作中、データ転送ばかりでなくバス状態の管理もほとんどターゲットが主導権を握っていて、イニシエータは受身になるのです。

(2) MSG, SEL, C/D, I/O 信号

これらの信号によって、フェーズと呼ばれるバスの状態が決まります。信号の組み合わせとそのときのバスの状態(フェーズ)を表 2 に示します。

表 2 のフェーズ中、セレクション・フェーズを除くすべてのフェーズは、ターゲットがこの四つの制御信号を操作することによって作られます。すなわち、ここでもターゲットに主導権があるわけです。それぞれのフェーズについては ② で詳しく述べます。

(3) BSY, ATN, RST 信号

これらの信号はそれぞれ独特な使われ方をしますので、別々に説明します。なお、ATN と RST は非同期条件信号といわれています。それは、この二つの信号が他の信号やそのときのフェーズに無関係に出力できるからです。

● BSY 信号

BSY 信号の使われ方には 3 種類あります。一つは一般的な使われ方で、ある二つの装置(イニシエータとターゲット)が SCSI バス上で接続されていてバスを使

用中であることを他の装置に知らせるために使われます。つまり、バスの「使用中」を示す「Busy」の名のとおりです。この場合、BSY はターゲットによって駆動されます。

二つめは、アービトレーション・フェーズという状態のときの使われ方です。アービトレーション・フェーズについては後で述べますが、このフェーズは SCSI バス上のバスを使用したい装置がすべて参加して、バスの使用权を争うフェーズです。このとき、BSY 信号は、この使用权争いに参加したすべての装置によって駆動(“偽”→“真”にすることをいう。規格書では、アサートする、と表記されている)されます。アクティブ Low のワイヤード・オア接続になっているため、このように使えるのです。

三つめは、セレクション・フェーズとリセレクション・フェーズ中の使われ方です。このフェーズについても後で述べますが、ひとつの SCSI 装置が自分の通信相手を選ぶフェーズです。このとき BSY 信号は、最初は選ぶほうの装置によって駆動され、つぎに選ばれたほうの装置がこの「選択」に対する応答として BSY 信号を駆動します。

● ATN 信号

この信号はイニシエータからターゲットにいつでも送ることのできる信号です。今まで何度か述べたように、アービトレーション、セレクション以外のバス線

表 2 MSG, SEL, C/D, I/O 信号とバス・フェーズ

MSG	SEL	C/D	I/O	状態(フェーズ名)	データ・ライン上の転送方向	備 考
×	1	×	0	セレクション	(I→T)	*
×	1	×	1	リセレクション	(I←T)	
0	0	0	0	データ・アウト	I→T	データ・フェーズ
0	0	0	1	データ・イン	I←T	
0	0	1	0	コマンド	I→T	コマンド・フェーズ
0	0	1	1	ステータス	I←T	ステータス・フェーズ
1	0	0	0	未定義		
1	0	0	1	未定義		
1	0	1	0	メッセージ・アウト	I→T	メッセージ・フェーズ
1	0	1	1	メッセージ・イン	I←T	

1 : 真, アクティブ Low なので実際の信号は Low(“L”)

0 : 偽, アクティブ Low なので実際の信号は High(“H”)

I : イニシエータ

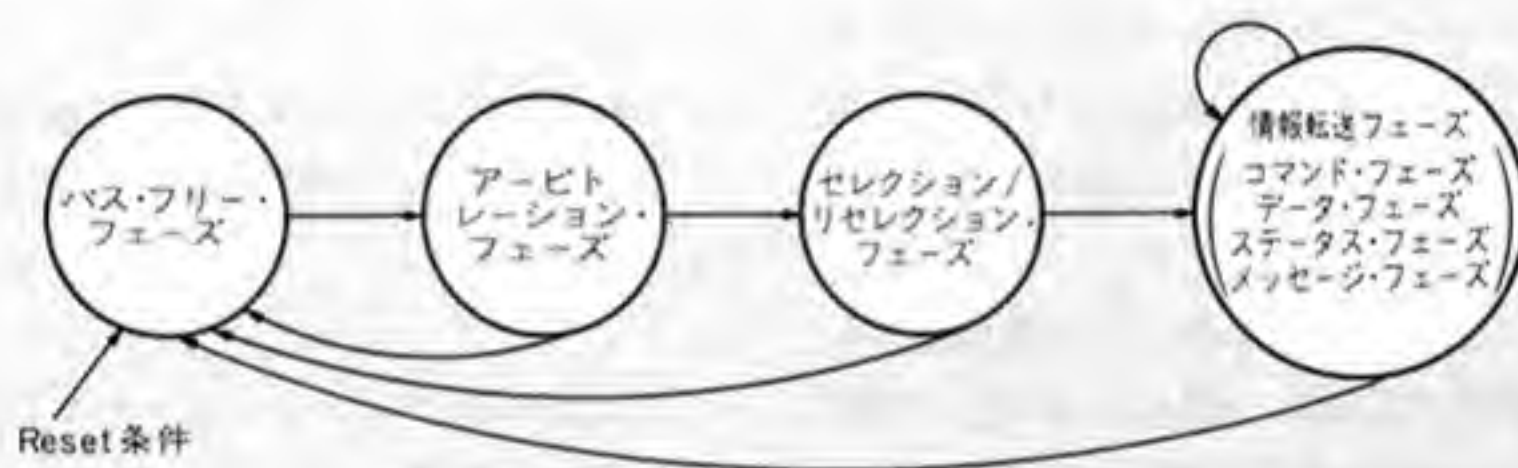
T : ターゲット

* : これらのフェーズでは REQ, ACK をともなった実際のデータ転送は行われない

表3 SCSI バス・フェーズ

フェーズ名	意 味・内 容	備 考
バス・フリー	バスを使用していない状態。	
アービトレーション	バスの使用権を各装置が取り合う。	
セレクション	イニシエータが使いたい装置(ターゲット)を選ぶ。	
リセレクション	ディスコネクト後、ターゲットがイニシエータと再度接続したいときに、ターゲットがイニシエータを指名する。	ディスコネクトについてはコラム(p.13)参照
コマンド	ターゲットがイニシエータからの命令を受け取る。	これら四つを総称して情報転送フェーズと呼ぶ。
データ	データを転送する。	
ステータス	ターゲットがイニシエータへコマンドの実行結果を知らせる。	
メッセージ	ターゲットとイニシエータ間での上記以外の情報連絡をする。	

図8 フェーズの変化



働中のフェーズでは、すべてターゲットがバス状態を管理しています。そこで、イニシエータがどうしてもターゲットに対し何かいいたい場合、この ATN を立てることによりそれが可能となります。

実際の「イニシエータが何かをいう」という行為はメッセージ・アウトという形で実現されます。つまり、ATN がきたことを確認したターゲットは、SCSI 規約にもとづいたタイミングでメッセージ・アウト・フェーズに入り、イニシエータのいいたいことを聞いてあげるわけです。

● RST 信号

RST はリセット信号で、すべての装置がいつでも送ることができます。RST 信号を検出したすべての装置は、すべての信号を解放、すなわちバスの使用をやめなければなりません。

ハード・リセットは、いわゆるリセットで、ハード・リセットの組み込まれた装置は RST 信号が入ると初期状態に戻ります。

ソフト・リセットが組み込まれた装置では、RST 信号が入ったとき、そのときの状態や実行中のコマンド、どこまで実行したかなどを記憶したままバスを解放するリセットです(詳細は2章を参照)。

2

SCSI バス・フェーズ

これまでの説明の中にもフェーズという言葉がところどころにありました。ここではフェーズについて、もうすこし詳しく説明します。

SCSI でいうフェーズとは、バスの使用状態のことで「今、バス上で何をしているか」を表します。フェーズは、表3に示すように大きく分けて8種類です。

SCSI バスの状態、すなわちフェーズの変化には決められた順序があります。これを図8に示します。それではフェーズごとに説明していきましょう。

■ バス・フリー・フェーズ

SCSIバスをどの装置も使っていない状態です。他のどのフェーズからもそのフェーズを終了後、バス・フリーに入ることができますし、また、リセット条件が入ればすぐにバス・フリーとなります。信号状態は、BSYとSELが“偽”（不平衡の場合は“H”となる、以下同様）であればこのフェーズとなります。

■ アービトレーション・フェーズ

SCSIバスの使用权を決めるフェーズで、すべての装置が参加できます。このフェーズにはバス・フリー・フェーズからしか入れません。具体的な動作はつぎのようになります。

- (1) これからバスを使おうとする装置はバスの状態がバス・フリーであることを確認したあと、BSYを駆動し、自分のID番号をデータ・ラインに出力する。ID番号はデータ・ラインのDB0～DB7に1対1で対応している。たとえばID3の装置がバスを使いたい場合、BSYとDB3を“真”（不平衡の場合は“L”となる）とする。
- (2) 他の装置もバスを使いたい場合、この使用权争いに参加できる。参加の方法は、最初にBSYが“真”になってから（誰かが使用权を主張しはじめてから） $1.8\mu\text{s}$ （バス・セット・ディレイ）以内にBSYと自分のIDを出す。
- (3) 最初にBSYが“真”になってから、 $2.4\mu\text{s}$ （アービトレーション・ディレイ）たったとき、アービトレーションに参加した各装置はデータ・ラインを検索し、自分より大きい番号のIDがあるかどうか調べる。自分が一番大きいIDであれば勝ったことになりSELを“真”（“L”）にしてセレクション・フェーズ、またはリセレクション・フェーズに入る。負けた装置は、自分のIDとBSYを“偽”（“H”）にして手を引く。

以上がアービトレーションの手順です。

■ セレクション・フェーズとリセレクション・フェーズ

アービトレーションに勝ったイニシエータは、これから使おうとするターゲットを選択します。

- (1) アービトレーションに勝ったイニシエータは、アービトレーション・フェーズを終わる前に少なくと

も $1.2\mu\text{s}$ の間BSYとSELを“真”（“L”）とする。これは、アービトレーションに参加したほかの装置が手を引くのを待つため。

- (2) セレクションはイニシエータからの出力フェーズなのでI/Oは“偽”（“H”）としておく。SELはもちろん“真”（“L”）にしておく。
- (3) イニシエータは自分自身のIDと選びたいターゲットのIDおよびATNを出力してからBSYを“偽”（“H”）にする。ATNを出力する理由は、このセレクション・フェーズの後にメッセージ・アウト・フェーズが続くことを表明するため。
- (4) 各装置はセレクション・フェーズ中、BSYが“偽”になったのを見て自分のIDが“真”となっているかどうかを見る。自分のIDが立っていた場合BSYを“真”（“L”）にして応答する。この応答はセレクション・アボート・タイム（ $200\mu\text{s}$ ）以内でなければならない。
- (5) イニシエータは、BSYによる応答が返ったことを確認したら、SELを解放しセレクション・フェーズを終わる。

以上がセレクション・フェーズですが、(4)で三つ以上のIDが“真”となっている場合、ターゲットはBSYによる応答をしません。選択されたターゲットは、続く一連のフェーズが終了するまでBSYを“真”（“L”）にしつづけ、バスを使用中であることを示しつづけます。

ここで、リセレクション・フェーズについても説明しておきます。このフェーズはオプションですが、非常に一般的な“必要な機能”となっていますので知っておくべきでしょう。

“リセレクション”とは、セレクションとは逆に、ターゲットがイニシエータを選ぶ行為で、なぜそんなことが？とお思いの方は「ディスコネクトとリコネクト」の項やコラムをざっと見てください。

リセレクション・フェーズに入る前に、ターゲットは当然、アービトレーションに参加し、バスの使用权を獲得しなければなりません。

- (1) アービトレーションに勝ったターゲットは、アービトレーションを終わる前に少なくとも $1.2\mu\text{s}$ の間BSYとSELを“真”とする。
- (2) 自分がターゲットであり、これはリセレクション・フェーズであることを主張するためI/Oを“真”

とする。

- (3) ターゲットは自分自身の ID と、選びたいイニシエータの ID を“真”とし、BSY を“偽”とし、イニシエータからの応答を待つ。
 - (4) SEL、I/O および自分の ID が“真”となっている（つまりリセレクトされた）ことを確認したイニシエータは、データ・バスを調べ、どのターゲットによって選ばれたかを確認し、応答として BSY を“真”にする。この応答はセレクション・アボート・タイム（200 μ s）以内でなければならない。
 - (5) BSY による応答を確認したターゲットは、やはり BSY を“真”に駆動し、その後 SEL を手放す（“偽”とする）。SEL が“偽”となったことを確認したイニシエータは BSY を手放す。ターゲットは BSY を“真”に保持したまま、続く一連のフェーズ（動作）を行う。
- 以上がリセクション・フェーズです。セレクション・フェーズとの違いがわかったでしょうか。

■ 情報転送フェーズ

情報転送フェーズには、① メッセージ、② コマンド、③ データ、④ ステータスの四つのフェーズがあります。ここでは、それぞれのフェーズに分けて説明します。

転送の共通事項として、このフェーズ中にターゲットにより駆動される I/O 信号によって転送方向が決

まります。I/O が“真”（“L”）のときはターゲットからイニシエータへ、I/O が“偽”（“H”）のときはイニシエータからターゲットへ転送されます。また、転送にはバス構成（p.8 のシーケンス A、B）で説明したように REQ/ACK によるハンドシェイク転送が行われる非同期転送（転送速度約 1.5 M バイト/秒まで）と、4~5 M バイト/秒までの同期転送、5~10 M バイト/秒まで可能な FAST SCSI と呼ばれる規約、およびバス幅を 2 バイトまたは 4 バイトまで広げた WIDE SCSI があります。

イニシエータ、ターゲット間でなんの約束事もなければ（デフォルトでは）非同期転送が行われますが、両者間でのメッセージのやりとりによって合意すれば、同期転送、FAST SCSI、WIDE SCSI のような高性能の転送が可能となります（2 章参照）。

[1] メッセージ・フェーズ

メッセージ・フェーズにはメッセージ・インとメッセージ・アウトの二つがあります。このフェーズに入るためには、ターゲットは MSG と C/D を“真”とし、I/O が“真”ならばメッセージ・イン（メッセージがターゲットからイニシエータへ）、“偽”ならばメッセージ・アウト（イニシエータからターゲットへ）となります。メッセージは 1 バイトまたは複数バイトによって構成されます。どんなメッセージがあるかは表 4 および表 5 を参照してください。SCSI 装置はこれら



なぜディスクコネク/リコネクなのか

ここで SCSI の一つの特徴であるディスクコネク/リコネクについて簡単に説明しておきます。イニシエータの命令をターゲットが実行する際、比較的時間のかかる動作もあります。たとえば、300 M バイトのディスク・ドライブを全部フォーマットする際、1 命令でターゲットは動作を始めますが、実行時間は 10 分ほどかかります。また、大量のデータをディスク・ドライブから読み出す命令を実行すると、途中でシーク動作（ヘッド位置の移動）が入る場合があります。この時間は 5 ms 程度ですが、この間のデータ読み出しはできないので 4 M バイト/秒で動作している SCSI バ

スにとっては比較的長い時間といえるでしょう。

このような場合にターゲットはいったんイニシエータとの接続を断ち切って（ディスクコネク）、バスを解放し、他の装置にバスを使用させることができます。そしてターゲットがコマンドを完了したり、再びデータの転送ができる状態になったとき、イニシエータに対して再接続（リコネク）を要求できます。

このような動作がディスクコネク/リコネクで、これにより、バスの使用効率を向上できます。これは、SCSI 規格上はオプションですが、スループットを上げるために現在では SCSI 装置必須の機能といえます。

表4 メッセージ一覧

メッセージ・コード	メッセージ名称	ATN 解除条件	方向	サポートの要否	
				I	T
00h	Command Complete：コマンドの終了を示す。	—	IN	M	M
01h	拡張メッセージ（表5）	—	—	—	—
02h	Save Data Pointer：現状のアクティブ・ポインタをセーブさせる。	—	IN	O	O
03h	Restore Pointer：最新のセーブされたポインタをアクティブ・ポインタに写す。	—	IN	O	O
04h	Disconnect： { IN の場合：ディスコネクトの通知。この後、バス・フリーとなる。 { OUT の場合：イニシエータからのディスコネクト要求。	—	IN	O	O
		YES	OUT	O	O
05h	Initiator Detected Error：ターゲットのリトライによっても回復不能なエラーの発生を通知。	YES	OUT	M	M
06h	Abort：そのイニシエータからのコマンド・キュー、実行中のコマンドなどをすべて中止する要求。この後、バス・フリーとなる。他のイニシエータのコマンドなどには影響しない。	YES	OUT	O	M
07h	Message Reject：最後に受け取ったメッセージが受け付けられないことを示す。正しく受け取れなかった、またはサポートしていないメッセージだった。	—	IN	M	M
		YES	OUT	M	M
08h	No Operation：メッセージ・イン・フェーズになったが（ターゲットにメッセージを要求されたが）送るべきメッセージがない場合に使用される。	YES	OUT	M	M
09h	Message Parity Error：最後のメッセージを受け取ったときにパリティ・エラーが発生していたことを示す。この後、ターゲットは、もう一度メッセージを送る。	YES	OUT	M	M
0Ah	Linked Command Complete：リンクされたコマンドの一つが終了したことを示す。イニシエータは続くコマンドのためにポインタを初期化する。	—	IN	O	O
0Bh	Linked Command Complete With Flag：フラグ付きのリンクされたコマンドが、終了されたことを示す。	—	IN	O	O
0Ch	Bus Device Reset：特定のターゲットをハード・リセットする。	YES	OUT	O	M
0Dh	Abort Tag：ある特定のコマンドの実行を中止させる。	YES	OUT	O	O
0Eh	Clear Queue：特定のロジカル・ユニットに対するすべてのイニシエータからのすべてのコマンドをクリアする。	YES	OUT	O	O
0Fh	Initiate Recovery：ECA (Extended Contingent Allegiance) 状態に入る通知。	—	IN	O	O
		YES	OUT	O	O
10h	Release Recovery：ECA 状態の解除を宣言する。この後、バス・フリーとなる。	YES	OUT	O	O
11h	Terminate I/O Process：実行中の I/O プロセスの終了を要望する。	YES	OUT	O	O
12h～1Fh	（1 バイト長メッセージ）	—	—	R	R
20h	Simple Queue Tag： { OUT の場合：コマンドにタグ番号を付ける。実行順序は問わない。 { IN の場合：ディスコネクトなどをする場合、どのコマンドかを知らせる。	—	IN	O	O
		NO	OUT	O	O
21h	Head of Queue Tag：このタグ付きコマンドは、コマンド・キューの先頭に入る。	NO	OUT	O	O
22h	Ordered Queue Tag：このタグ付きコマンドは到着した順に実行される。	NO	OUT	O	O
23h	Ignore Wide Residue：WIDE SCSI の場合、最終データの不要（無視すべき）バイト数を知らせる。	—	IN	O	O
24h～2Fh	（2 バイト長メッセージ）	—	—	R	R
30h～7Fh		—	—	R	R
80h～FFh	Identify：ロジカル・ユニット番号の指定や、ディスコネクトの可/不可を示す。	—	IN	M	M
		NO	OUT	M	M

I：イニシエータ，T：ターゲット

M：サポート必須，O：オプション，R：リザーブ

IN：ターゲット→イニシエータ，OUT：イニシエータ→ターゲット

表中の ATN 解除条件の欄が“YES”のメッセージをイニシエータが送信するとき、イニシエータはそのメッセージ・アウト・フェーズの最後の ACK 信号応答の 90 ns (Deskew Delay×2) 以上前に ATN 信号を False にし、アテンション・コンディションを解除しなければならない。この条件に違反したとき、ターゲットはプロトコル上のエラーとみなしてバス・フリー・フェーズに移行する。

表5 拡張メッセージ一覧

拡張メッセージ・コード	メッセージ長 (コード)	メッセージ名称	ATN 解除条件	方向	サポートの要否	
					I	T
00h	7	Modify Data Pointer：現状のデータ・ポインタ値の変更要求。	—	IN	O	O
01h	5	Synchronous Data Transfer Request：データの同期転送要求。転送周期、オフセット値も入っている。	—	IN	O	O
			YES	OUT	O	O
02h	—	(Extended Identify)⇒削除	—	—	R	R
03h	4	Wide Data Transfer Request：ワイド・データ転送要求。データ・バス幅が入る。	—	IN	O	O
			YES	OUT	O	O
04h～7Fh	—		—	—	R	R
80h～FFh	—		—	—	V	V

I：イニシエータ、T：ターゲット

O：オプション、V：ベンダ固有、R：リザーブ

IN：ターゲット→イニシエータ、OUT：イニシエータ→ターゲット

ATN：解除条件：表4

図9 Identify メッセージ

ビット	7	6	5	4	3	2	1	0
バイト0	1	DiscPriv	LUNTRN	0	0	LUNTRN		

↑
Identify メッセージであることを示す

LUNTRN	LUNTRN
0	ロジカル・ユニット番号 (LUN)
1	ターゲット・ルーチン番号

のメッセージのやりとりによって、装置間の物理的あるいは論理的な約束事を取り決めます。すべてを詳細に説明すると、頭が混乱するといけませんので、いくつかの重要なメッセージをとりあげ説明します。

● 00h コマンド・コンプリート

ターゲットからイニシエータへ送られるメッセージで、イニシエータからの命令(コマンド)が終了したことを意味します。

● 04h ディスコネクト・メッセージ

このメッセージは通常、ターゲットが発行するもので、ターゲットがイニシエータの命令を実行するのに長時間かかるような場合、イニシエータとの接続をいったん切り離し、バスを他の装置が使えるようにするものです。ターゲットがこのメッセージを出す場合、つぎに説明するセーブ・データ・ポインタ・メッセージと対になって使われます。

イニシエータがターゲットに対しこのディスコネク

ト・メッセージを発行した場合、ターゲットはバスを切り離し、バス・フリー・フェーズに入らなければなりません。

● 02h セーブ・データ・ポインタ・メッセージ

これはターゲットが発行するもので、ターゲットがディスコネクト・メッセージを発行する前にターゲットからイニシエータに送られます。これによりイニシエータはどんなコマンドを、どこまで実行したか(例えば長いデータを要求し、どこまで送られたかなど)を覚えておくことができるわけです。

● 80h～FFh アイデンティファイ・メッセージ

セレクション・フェーズのあとイニシエータからターゲットへ必ず送られるメッセージで、図9のようになっています。ビット0～2は通常ターゲットの下のロジカル・ユニット番号の指定に使われます。ビット6の DiscPriv (Disconnect Privilege：ディスコネクトの特典付き)は、“1”であればターゲットがコマンド

実行中にバスを切り離す(ディスコネクト)ことができることを意味します。

このメッセージはリセクション・フェーズに続いてターゲットからイニシエータにも送られます。

● 01h+03h+01h+mh+xh シンクロナス・データ・トランスファ・リクエスト・メッセージ(同期転送要求)

このメッセージのやりとりによって、イニシエータとターゲットは高速のデータ転送を行うことができるようになります。メッセージのなかに、何バイト(*mh*)分のオフセット(データの先送り)が許されるか、また、どのくらいの転送速度でデータ転送を行うかの数値(*xh*)が含まれます。

● コマンド・キューイング用メッセージ

表4のなかに、○○ Queueとか○○ Tagというメッセージがありますが、これらはすべてコマンド・キューイング用のメッセージです。コマンド・キューイングとは、あるイニシエータからあるターゲットへ複数のコマンドを発行でき、ターゲットはそれらのコマンドをいったんためこみ、効率のよい順番で実行し

たり、あるいは、イニシエータに指定された順番でコマンドを実行して行きます。これによりホスト・コンピュータのCPUの効率的使用やバスの有効利用がさらにはかれる可能性があります。

[2] コマンド・フェーズ

コマンド・フェーズはその名のとおりイニシエータからターゲットへコマンド(命令)を送るフェーズです。ターゲットはイニシエータからコマンドを受け取るため、MSG, I/Oを“偽”(“H”), C/Dを“真”(“L”)としてこのフェーズへ入り、REQを“真”(“L”)としてコマンドの1バイト目(バイト番号0)を待ちます。そのあとは、前に述べたようにREQ/ACKのハンドシェイクによってコマンド全部の転送が行われます。コマンドの形式はグループによって6バイト・コマンド、10バイト・コマンド、12バイト・コマンドなどがあります(図10, 11, 12)。

8種類(3ビット)のコマンド・グループに対し5ビットのコマンド・コードが存在するので、合計256個の命令が構成できます。グループ・コードはコマンドの1バイト目に含まれていて、ターゲットはここで何

図10 グループ0のCDB基本形式

の CDB 基本形式

バイト \ ビット	7	6	5	4	3	2	1	0
0	0	0	0	グループ・コード				
1	LUN			論理ブロック・アドレス (MSB)				
2	論理ブロック・アドレス							
3	論理ブロック・アドレス (LSB)							
4	転送データ長							
5	(V)	(R)					フラグ	リンク

オペレーション・コード
(OP コード)

コントロール・バイト

(V) : ペンタ・ユニーク
(R) : リザーブ (予約)

(V):ペンタ・ユニーク
(R):リザーブ(予約)

図11 グループ1および2のCDB基本形式

ビット バイト	7	6	5	4	3	2	1	0
0	グループ・コード*			コマンド・コード				
1	LUN			(R)				RelAdr
2	論理ブロック・アドレス (MSB)							
3	論理ブロック・アドレス							
4	論理ブロック・アドレス							
5	論理ブロック・アドレス (LSB)							
6	(R)							
7	転送データ長 (MSB)							
8	転送データ長 (LSB)							
9	(V)		(R)				フラグ	リンク

* 001 または 010

図12 グループ5のCDB基本形式

バイト \ ビット	7	6	5	4	3	2	1	0	
0	1	0	1	グループ・コード コマンド・コード					
1	LUN			(R)				RelAdr	
2	論理ブロック・アドレス (MSB)								
3	論理ブロック・アドレス								
4	論理ブロック・アドレス								
5	論理ブロック・アドレス (LSB)								
6	(R)								
7	(R)								
8	(R)								
9	転送データ長 (MSB)								
10	転送データ長 (LSB)								
11	(V)			(R)				フラグ	リンク

バイトの命令がくるか判断できるのです。

図10のようなコマンド・コードとそれにつづく情報を CDB(Command Descriptor Block)といいます。コマンドは必ずこの形で送られます。

わかりやすい一例を上げると、たとえばコマンド・コード(0バイト目)が08:リード・コマンドで、論理ブロック・アドレスが“0”, 転送データ長が“2”, の場合, “0”番地から2ブロック分のデータを読み出し、イニシエータへ送れ、という意味になります。コマンドの種類によっては、論理ブロック・アドレスが不要であったり、転送データ長のところが、パラメータの長さ(バイト長)であったりもします。実際にコマンドを組む際は、装置ごとのマニュアルをコマンドごとによく見て組んでください。前にもふれましたが、CDBのなかの“LUN”は現実にはあまり意味をもっていません。SCSI-2では“0”にすることを推奨しています。なぜならばメッセージ・フェーズのところで説明した、Identify メッセージの“LUN”でロジカル・ユニット番号が確定するからです

[3] データ・フェーズ

ターゲットはMSG, C/Dを“偽”(“H”)としてデータ・フェーズに入り、I/Oが“真”か“偽”かによってデータの転送方向がターゲットからイニシエータ、イ

ニシエータからターゲットとなります。

転送はREQ/ACKによるハンドシェイクで行われます。転送されるデータ長(データ量)はCDBの中で決められています。

[4] ステータス・フェーズ

ステータス・フェーズは、ターゲットがイニシエータにステータス情報を送る場合に使うフェーズで、MSGを“偽”(“H”), C/DとI/Oを“真”(“L”)にすることによりこのフェーズに入ります。

ステータス情報というのは、イニシエータから受け取ったコマンドの実行結果をイニシエータに知らせるもので、1バイトのデータです。たとえばコマンドを正常終了した場合、00h(Good ステータス)を、また異常終了つまりリード・エラーがあった場合などは、02h(Check Condition)をイニシエータに送ります。

ステータスの一覧を図13に示します。

3

ディスコネクトとリコネクト

SCSIバスには基本的にたくさんの装置が接続されますが、実際にバスを使えるのはアービトレーション

図13 ステータス・バイト

ビット	7	6	5	4	3	2	1	0
バイト0	R	R	ステータス・バイト・コード					R

R: リザーブ (=0)

ビット 6, 5, 0 はベンダ固有に定義できなくなった

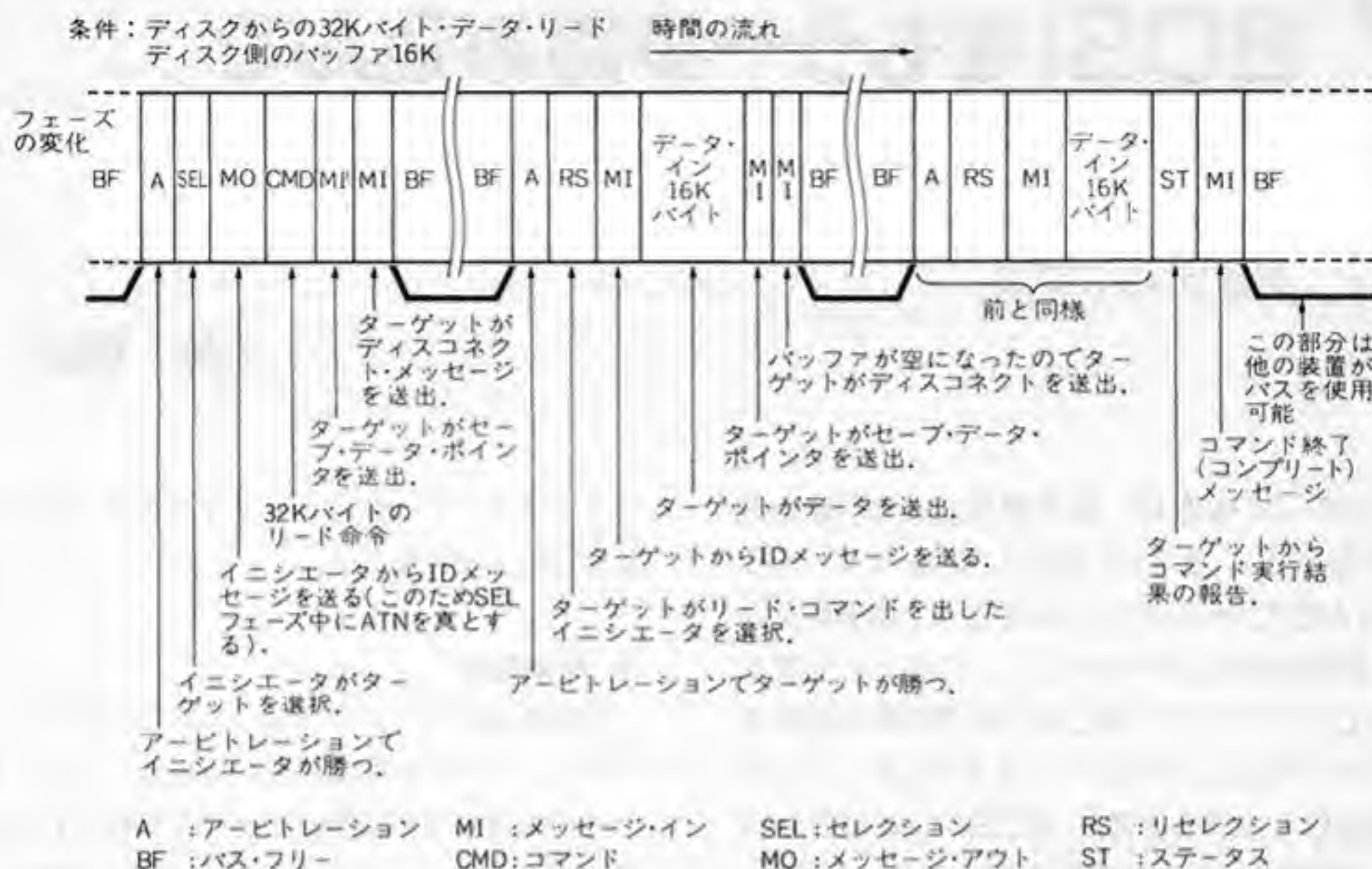
ビット	54321	
00000		Good
00001		Check Condition
00010		Condition Met
00100		Busy
01000		Intermediate
01010		Intermediate-Condition Met
01100		Reservation Conflict
10001		Command Terminated
10100		Queue Full

イニシエータ（ホスト）

ターゲット(HDD)

*データの転送は1バイトごとに、REQ/ACKのハンドシェイクによって転送される

図14 ディスコネクト/リコネクトによるバスの使用状況例



に勝った装置と、その装置に選ばれた装置、すなわち同時には一対のイニシエータとターゲットしか使えません。

コラムにもあるようにフォーマット・ユニットというたった6バイトの命令で実行時間が10分もかかる場合、「ターゲットはBSYを立てたまま」「イニシエータはGoodステータスとCommand Completeメッセージを待っているだけ」でよいのでしょうか、その間、もしバスを解放できれば他の装置が、仕事ができるのです。こういった考えから生まれたのが、ディスコネクト/リコネクトのオプションです。

ディスコネクトするかどうかはターゲットが判断するもので、装置の設計思想、実行中のコマンドなどによりディスコネクトするための条件は違ってきます。たとえば、カートリッジ・テープ・ドライブでLoad命令を実行すると、ドライブの実際の動作はテープの最初の部分を探すわけですからテープ走行という時間の長い動作が必要なので、カートリッジ・テープ・ドライブは、Load命令を受け取るとすぐにディスコネクトしバスを解放して動作が終わったときにリコネクト(再接続)の要求を出します。

また、ディスク・ドライブからデータを読み出す命令(Read命令)の場合、データ・イン・フェーズ中ディ

スク・ドライブ上のデータ・バッファが空になるといったんディスコネクトし、バッファがいっぱいになるまでディスク面上のデータをバッファに入れてからリコネクトして、データ転送の続きを行ったりします。この例を図14に示します。この図は一般的なディスコネクトの例ですが、通常の動作状態におけるSCSIのフェーズをすべて含んでいるので参考になると思います。

いままでの説明をReadコマンドの実行例として表6にまとめます。一つのReadコマンドを実行するために、SCSIは複雑な動きをします。しかし、整理してみれば、SCSIはとても論理的なプロトコルであることがわかるでしょう。

■ むすび

ここでは、SCSIの基本的な部分のみを説明しました。基本がある程度わかれば、この先いろいろむずかしい機能が出てきても理解しやすいと思います。SCSIも-1から-2になって機能が飛躍的に増えたので「スミからスミまで読む」というしろものではなくなくなってしまったようです。「必要に応じて、規格書を見る」のが正しい方法でしょう。

おおしま・ひろたか タンベルグデータ株式会社営業技術本部

SCSIをもう一步踏み込んで理解する

データ転送/バス条件/SCSIポインタ/メッセージ・システム/ステータス/
共通コマンド詳説

大島 啓孝

SCSIのデータ転送は、非同期転送と同期転送(FAST SCSI)、8ビット転送と16/32ビットのワイド転送(WIDE SCSI)に大別できる。基本は非同期転送。同期転送は、ターゲット-イニシエータ間でREQ/ACK オフセット値と最小転送周期をきめる必要がある。WIDE SCSIは、基本のAケーブル以外に拡張用Bケーブルを使う。SCSIには、送りたいメッセージがあることを知らせる「アテンション条件」と、バスをリセットさせる「リセット条件」があり、これによって、バス・フェーズが変わる。この章では、以上の動きを克明に追ったあと、SCSIの動作で欠かせないSCSIポインタとメッセージ・システム、それにステータスについてふれる。そして、最後に全デバイス共通コマンドについて整理する。(編集部)

1

データ転送 (FAST SCSI, WIDE SCSI)

SCSI上のデータ転送方式は、転送速度にかかわる区別では、基本となる非同期転送と、高速化が可能な同期転送(FAST SCSIはこの同期転送の一部)、転送のデータ幅による区別では、通常の8ビット(1バイト)幅の転送と、16ビットまたは32ビット(2バイトまたは4バイト幅)のワイド転送(WIDE SCSI)というように分類できます。

■ 非同期転送

この転送方法は基本中の基本で、データ転送以外の情報転送(メッセージ、ステータス)もこの方法で送られます。具体的には1章の「シーケンスA」, 「シーケンスB」のと通りのREQ/ACKによるハンドシェイク

によります。この転送モードでは約1.5 Mバイト/秒程度の転送が可能です。

■ 同期転送

同期転送はデータ・フェーズでのみ使用できます。つまり、データの高速度転送を目的としたモードです。データ転送にこの同期転送モードを利用するためには、ターゲット-イニシエータ間での同意が必要です。この同意は、Synchronous Data Transfer Request (SDTR)というメッセージのやりとりによって「REQ/ACK オフセット値」と「最小転送同期(Minimum Transfer Period)」の二つの値をとり決めることによって行われます(④の「メッセージ・システムの詳細」参照)。

REQ/ACK オフセット値とは、ACKパルスを受け取る前に先行して送ることのできるREQパルスの数のことです。これによりターゲットは、データを送る場合、ACKの応答を待たずにデータの先出しができ、データを受け取る場合はACKの応答を待たずに準備できること、つまりデータ要求ができるわけです。

非同期転送モードでは一つのREQパルスに対して一つのACKパルスの応答を待たなければいけないので、信号の「行って来る時間」がつかねに必要ですが、同期転送モードでは応答時間分は待たなくてよいわけです。

最小転送周期はREQとACKの両方を規定する値で、「REQ/ACKをこの値未満の時間で繰り返してはいけない」という同意事項です。最小の設定可能な値は100 ns(ナノ秒)すなわち転送速度は1バイト・データ・バスで1/100 ns=10 Mバイト/秒となっています。SDTRメッセージの中では値“1”は4 nsで1バイトのパラメータなので、25から255(100~1020 ns)で設定

が可能です。

■ FAST SCSI

SCSI のタイミング規定(表1)の中には“FAST ○○”という名称で定義されているタイミングがあります。これらの値が適用されるのは、SDTR のやりとりによりターゲットとイニシエータが「最小転送周期」200 ns 未満(1 バイト・データ・バスで5 M バイト/秒を超える)で同意した場合で、これが FAST SCSI と呼ばれています。

■ WIDE SCSI

ワイド転送も同期転送と同様にデータ・フェーズのみで使用可能なモードで、ターゲット-イニシエータ間の Wide Data Transfer Request (WDTR) メッセージのやりとりにより確立されます。物理的には、標準の A ケーブル(50 ピン)とは別の拡張用 B ケーブルを使用し、A ケーブル上の1 バイトと並行して、B ケーブル上で1 バイトまたは3 バイト、合計2 バイトまたは4 バイトのデータを転送する方式です。

2 バイト転送か4 バイト転送かは、WDTR メッセージの交換によって同意されますが、4 バイト転送をサポートしている装置は2 バイト転送もサポートできるように推奨されています。データの転送順序は、先頭バイトを A ケーブルに、続くデータ・バイトを B ケーブルにのせることになっています。

あるコマンドに対する最後の転送で、全バイトを使用する必要がない場合でも、不要バイト上はどんなデ

ータでもかまいませんが、“正しく”(パリティのセットなど)送れなくてはなりません。これらの不要バイトは、後で Ignore Wide Residue というメッセージによって無視されます。

転送タイミングは、A ケーブル上は REQ/ACK で、B ケーブル上は REQB/ACKB という信号で別々に制御されます。これにより A ケーブルと B ケーブルの長さや、接続されている他の装置の影響による伝播遅延などを考慮せずにすむわけです(図1)。

当然のことながら、A/B 両ケーブルは同一の転送モード(同期または非同期)で動作させなくてはなりません。

2

SCSI バス・コンディション (バス条件)

SCSI バスには「アテンション・コンディション(条件)」と「リセット・コンディション(条件)」という二つの非同期条件(通常のフェーズの流れを変えさせる条件)があります。これらの条件を検出した場合、SCSI 装置はそれに応じた動きをするよう規定されています。

■ アテンション条件

これはイニシエータがターゲットに対し、送りたいメッセージがあることを表明する条件で、具体的にはイニシエータが ATN 信号を“真”にすることによ

SCSI のデータ転送方式の分類

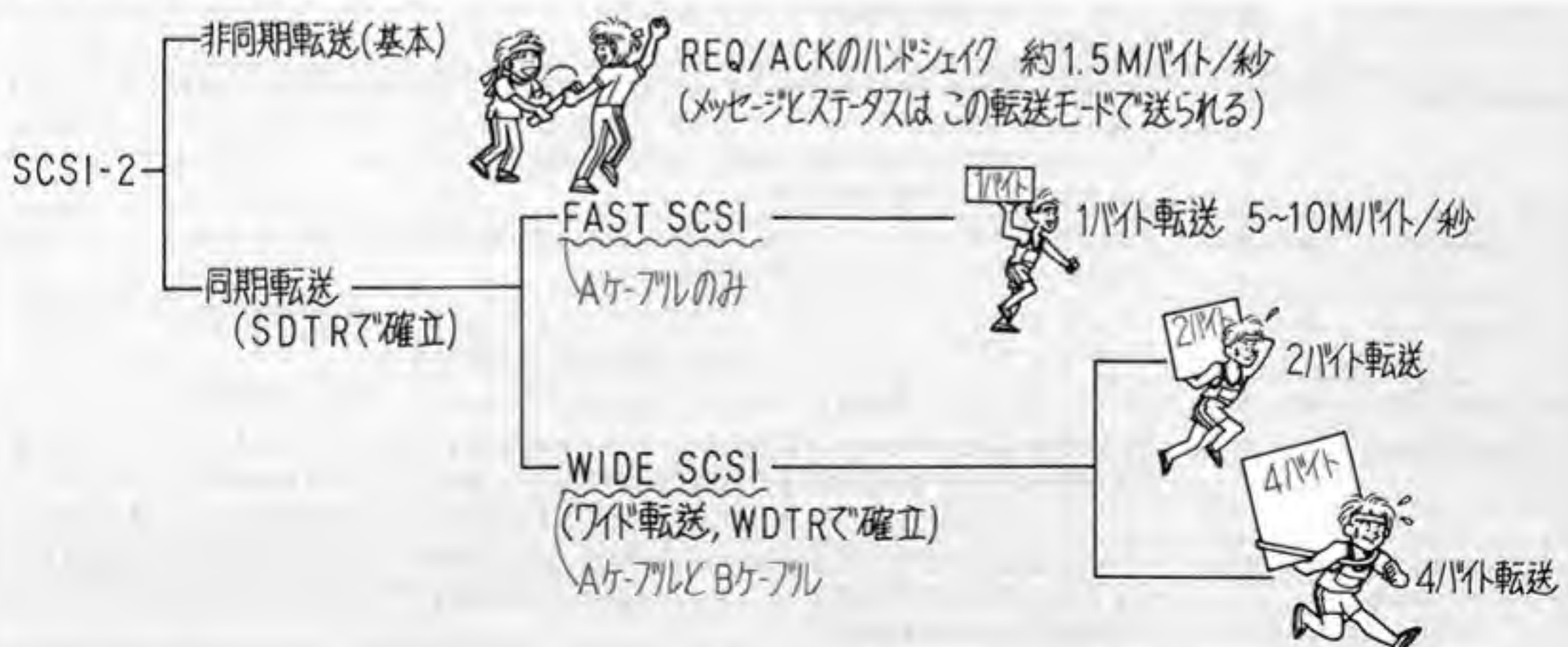
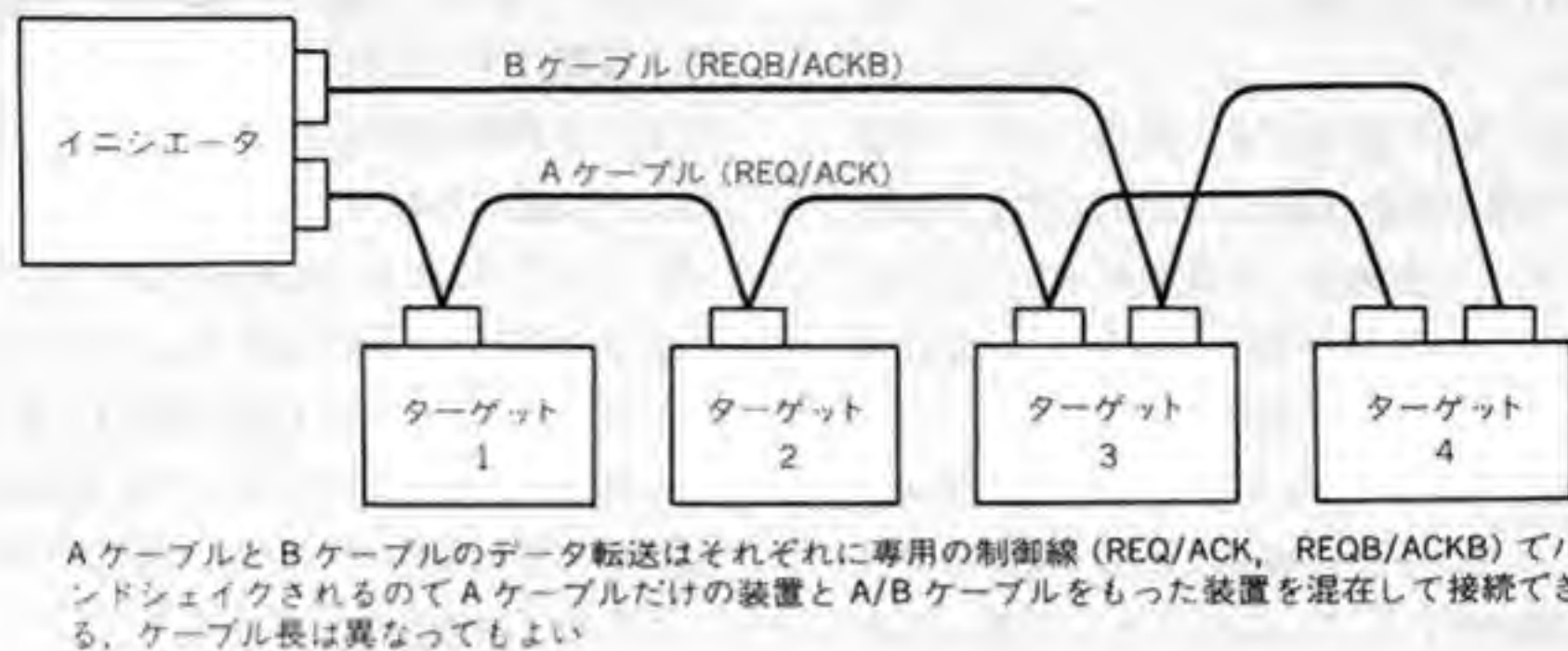


表1 SCSIのタイミング規定

名 称	規定値	タイミング規定
Arbitration Delay	最小 2.4 μ s	アービトレーション・フェーズで SCSI デバイスが BSY 信号および SCSI ID を送出してからバス使用権の優先順位を決定するためにデータ・バス上の値を判定するまでの最小待機時間、最大待機時間は規定されない
Assertion Period	最小 90 ns	同期モードのデータ転送時における REQ 信号および ACK 信号(B ケーブルについては REQB 信号および ACKB 信号)の最小パルス幅
Bus Clear Delay	最大 800 ns	以下のいずれかの事象が発生してから SCSI デバイスがすべてのバス信号の駆動を停止するまでの最大許容時間 ① バス・フリー・フェーズを検出したとき ② アービトレーション・フェーズを実行中に他の SCSI デバイスが SEL 信号を“真”にしたことを検出したとき ③ RST 信号が“真”になったこと(リセット・コンディション)を検出したとき
Bus Free Delay	最小 800 ns	SCSI デバイスがバス・フリー・フェーズを検出してからアービトレーション・フェーズを開始するために BSY 信号および SCSI ID を送出するまでの最小待機時間
Bus Set Delay	最大 1.8 μ s	SCSI デバイスがバス・フリー・フェーズを検出してからアービトレーション・フェーズを開始するために BSY 信号と SCSI ID とを送出するまでの最大許容時間
Bus Settle Delay	最小 400 ns	特定の制御信号の状態が変化してからバスの状態が安定するまでの最小待機時間
Cable Skew Delay	最大 10 ns	任意の 2 台の SCSI デバイス間で任意のバス信号間のインターフェース・ケーブル上の信号伝播時間の最大許容差
Data Release Delay	最大 400 ns	I/O 信号の状態が“偽”から“真”に変化した後、イニシエータがデータ・バス信号の駆動を停止するまでの最大許容時間
Deskew Delay	最小 45 ns	バス信号間の伝送スキュー補償の最小時間
Disconnection Delay	最小 200 μ s	ターゲットがイニシエータからの Disconnect メッセージによりバス・フリー・フェーズに移行した後、つぎにアービトレーションに参加するまでの最小待機時間
Hold Time	最小 45 ns	同期モードのデータ転送においてデータを受信する SCSI デバイスでのホールド時間を補償するために REQ 信号または ACK 信号(B ケーブルについては, REQB 信号と ACKB 信号)パルスの前縁からデータ・バス上の転送データを保持していなければならない最小時間
Negation Period	最小 90 ns	同期モードのデータ転送において REQ 信号の後縁からつぎの REQ 信号の前縁まで、または ACK 信号の後縁からつぎの ACK 信号の前縁まで(B ケーブルでは REQB 信号と ACKB 信号)の最小時間
Power-On to Selection Time	最大 10 sec [推奨値]	電源投入後にターゲットが Test Unit Ready, Inquiry または Request Sense コマンドに回答できるようになるまでの最大許容時間
Reset Hold Time	最小 25 μ s	リセット・コンディションを生成するとき RST 信号を“真”にする最小保持時間、最大時間は規定されない
Reset to Selection Time	最大 10 sec [推奨値]	ハード・リセット・コンディションが解除された後にターゲットが Test Unit Ready, Inquiry または Request Sense コマンドに回答できるようになるまでの最大許容時間
Selection Abort Time	最大 200 μ s	セレクション・フェーズまたはリセレクション・フェーズで SCSI デバイスが、自己が選択されていることを認識してから、BSY 信号を応答するまでの最大許容時間
Selection Timeout Delay	最小 250 ms [推奨値]	セレクション・フェーズまたはリセレクション・フェーズでイニシエータまたはターゲットがタイムアウト処理を開始するまでに選択対象の SCSI デバイスからの BSY 信号応答を待つ最小時間
Transfer Period	100 ns～ 1020 ns	同期モードのデータ転送において REQ 信号の前縁からつぎの REQ 信号の前縁まで、または ACK 信号の前縁からつぎの ACK 信号の前縁まで(B ケーブルでは REQB 信号と ACKB 信号)の最小時間[最小繰返し時間]、具体的な値は SDTR メッセージの交換により、イニシエータとターゲット間で決定される
Fast Assertion Period	最小 30 ns	FAST 同期モードのデータ転送時における REQ 信号および ACK 信号(B ケーブルでは REQB 信号および ACKB 信号)の最小パルス幅
Fast Cable Skew Delay	最大 5 ns	FAST 同期モード転送を行うときの任意の 2 台の SCSI デバイス間での任意のバス信号間のインターフェース・ケーブル上の信号伝播時間の最大許容差
Fast Deskew Delay	最小 20 ns	FAST 同期モード転送を行うときのバス信号間の伝送スキュー補償の最小時間
Fast Hold Time	最小 10 ns	FAST 同期モードのデータ転送においてデータを受信する SCSI デバイスでのホールド時間を補償するため、REQ 信号または ACK 信号(B ケーブルについては REQB 信号と ACKB 信号)パルスの前縁からデータ・バス上の転送データを保持していなければならない最小時間
Fast Negation Period	最小 30 ns	FAST 同期モードのデータ転送において REQ 信号の後縁からつぎの REQ 信号の前縁まで、または ACK 信号の後縁からつぎの ACK 信号の前縁まで(B ケーブルでは REQB 信号と ACKB 信号)の最小時間

図1 AケーブルとA/Bケーブル(WIDE SCSI)の混在接続



てこの条件に入ります。ここでイニシエータはメッセージ・アウト・フェーズを期待するわけです。ATNを“真”にすることのできるフェーズはアービトレーションとバス・フリー以外のフェーズでなければなりません。

ATNを出すタイミングは、あるフェーズ中、最後のバイトを転送するためのACKを落とす前(すなわち、つぎのフェーズに入る前)に出さなければ、つぎのフェーズ移行のときにメッセージ・アウト・フェーズにしてもらえない結果となります。また、ATN解除条件“YES”(1章の表4)となっているメッセージをイニシエータが送出したときは、そのメッセージ送出のためのACKを立てる(“真”にする)前にATNを落として(“偽”にして)おかなくてはなりません。イニシエータがこの条件を守らなかった場合、ターゲットはバス・フリー・フェーズに入ってしまいます(Unexpected Disconnect)。

ATN条件に対し、ターゲットが“メッセージ・アウト・フェーズ”に入るタイミングは以下のとおりです。

- (1) コマンド・フェーズ中にATNが“真”になった場合、CDB(Command Descriptor Block)バイト群の一部または全部を転送した後にメッセージ・アウト・フェーズに入る。
- (2) データ・フェーズ中にATNが“真”になった場合、“もっとも早い、都合のいいとき(ブロック境界である必要はない)”にメッセージ・アウト・フェーズに入る。イニシエータは、フェーズが変わるまで、データ転送のためのREQ/ACKハンドシェイクを続けなければならない。
- (3) ステータス・フェーズ中にATNが“真”になっ

た場合、ステータス・バイトがイニシエータに受け取られた後、メッセージ・アウト・フェーズに入る。

- (4) メッセージ・イン・フェーズ中にATNが“真”となった場合、他の(つぎの)メッセージを送る前にメッセージ・アウト・フェーズに入る。これによりメッセージ・パリティ・エラーが意味をもつようになる。

- (5) セレクション・フェーズで、イニシエータがBSY信号を手放す前にATNが“真”となった場合、そのセレクション・フェーズの直後にメッセージ・アウト・フェーズに入る。

- (6) リセレクション・フェーズ中にATNが“真”となった場合ターゲットは、アイデンティファイ・メッセージを送った後、メッセージ・アウト・フェーズに入る。

イニシエータもATN信号を以下のようにあつかわなければなりません。

- ▶ 2バイト以上のメッセージを送りたい場合、ATNを立てつづけなければならない。
- ▶ メッセージ・アウト・フェーズ中、ACKを“真”としている間以外であれば、いつでもATNを落とす(“偽”とする)ことができる。

通常、メッセージ・アウト中の最後のREQ/ACKハンドシェイク中、REQが“真”、ACKが“偽”である間にATNを“偽”とすることになっています。

■ リセット条件

リセット条件は、バスをリセット、すなわちバスに接続されているすべての装置に対し、即座にバスから手をひかせたいときに使われます。バス上がどんな状

態/フェーズであっても最優先の条件です。バス上の装置はだれでも RST 信号を“真”にすることによりこの条件が作れます。

この条件、つまり RST 信号がきた場合、すべての装置はすべての信号線を解放状態にしなければなりません(もちろん、リセットを発生した装置の RST 信号は含まれない)。リセット条件が発生したとき、「実行中であった作業」や「それ以前に設定された動作モード」や「リザーブ状態」などがどうなるかは、それぞれの装置が「ハード・リセット」の機能しかないか、「ソフト・リセット」機能をもっているかによって違ってきます。リセット条件後のフェーズは必ずバス・フリーです。ハード/ソフト両リセットの内容を説明しましょう。

(1) ハード・リセット

ハード・リセットは、一般的にいわれる「リセット」で、装置はすべての状態をクリアし、パワー・オン時のような初期状態になります。SCSI の規約では以下のように述べられています。

- ① すべての(キューに納められたものも含む)I/O プロセス(実行中または実行しなくてはならないイニシエータ/ターゲット間の動作)をクリアする。
- ② SCSI デバイスのリザーブ状態を解除する。
- ③ すべての動作モードを、通常のパワー・オン時のような「初期状態」にもどす。
- ④ ユニット・アテンション状態を設定する(コラム参照)。



I/O プロセス

イニシエータ・ターゲット間の 1 回の論理的な接続、つまりセレクションから始まり、単一コマンドまたはリンクド・コマンド・グループ(いくつかのリンクされたコマンド)の受け渡しを行い、終了するまでの一連のプロセスが一つの I/O プロセスです。一

つの I/O プロセス中には複数回のディスコネクト/リコネクトを含んだり、また終了の形もコマンド・コンプリートによる正常終了ばかりでなく、リセットなどによる異常終了のこともあります。

ユニット・アテンション状態

「ユニット・アテンション」はエラーなどの概要を表すセンス・キーの一つで、リクエスト・センス・コマンドに対する返事のセンス・データの中にあるコード名の一つです。

ユニット・アテンションの意味は、ターゲットがイニシエータに対し、「私を使うのに注意してください」ということで、たとえばパワー・オン直後やハードウェア・リセット直後にはターゲット内に必ず発生する状態です。このほかにもメディア(たとえばテープ装置のテープ)が交換されたり、いろいろなパラメータの入っているメモリの内容が書き換えられたりしたときにも発生します。この状態はターゲットの内部的な状態で、この状態に入ってから、イニシエータから Inquiry 以外のコマンドがきた場合ターゲットは(通常)Check Condition というステータスで答え、初めてユニット・アテンション状態であることを表明するチャンスが与えられます。イニシ

エータが Check Condition ステータスに対し、うまく(普通はこうなる)、Request Sense コマンドを出してくれれば、センス・データ内のセンス・キーによりユニット・アテンションを伝え、かつこの状態をクリアすることができるのですが、Request Sense 以外のコマンドでは、ただたんに状態をクリアし、そのコマンドを実行してしまいます(センス・データが刈り取られなかった)。

Inquiry コマンドはちょっと例外で、ユニット・アテンション状態に入ってから、1 番目のコマンドが Inquiry であった場合、この状態を保持したまま正常に実行/終了します。もし、他のコマンドが先に発行され、Check Condition で応答しているのに Inquiry が発行された場合は、ユニット・アテンション状態は他のコマンドのときと同様にクリアされてしまいます。

(2) ソフト・リセット

「ソフト・リセット」とは、そもそもマルチイニシエータ環境で、ある1台のイニシエータが「リセット」を出したときに、他のイニシエータそれ自身の動作や、それらのイニシエータからターゲットに対し発行済み/実行中のコマンドなどにできるだけ影響を与えずに、「リセット」を実施することが目的で作られました。ディスクコネク中のプロセスは、リセット後も平然と続けられるようにするためです。SCSI 規約では、

- ① 完全に認識された、終了していないすべての I/O プロセスを終了するように試みる。
- ② SCSI デバイスのリザーブ状態は保持する。
- ③ すべての動作モードを保持する。
- ④ リセット前にキューに入った I/O プロセスを実行するためのすべての必要な情報を保持する。

SCSI の規格書には、リセット発生時のプロセスの「認識」や「終了」をターゲット/イニシエータがそれぞれどう解釈すべきか、とか、データ・ポインタが正しくセーブされているかどうかをどのように理解するかなどのこまかいことも述べられています(規格書を参照)。

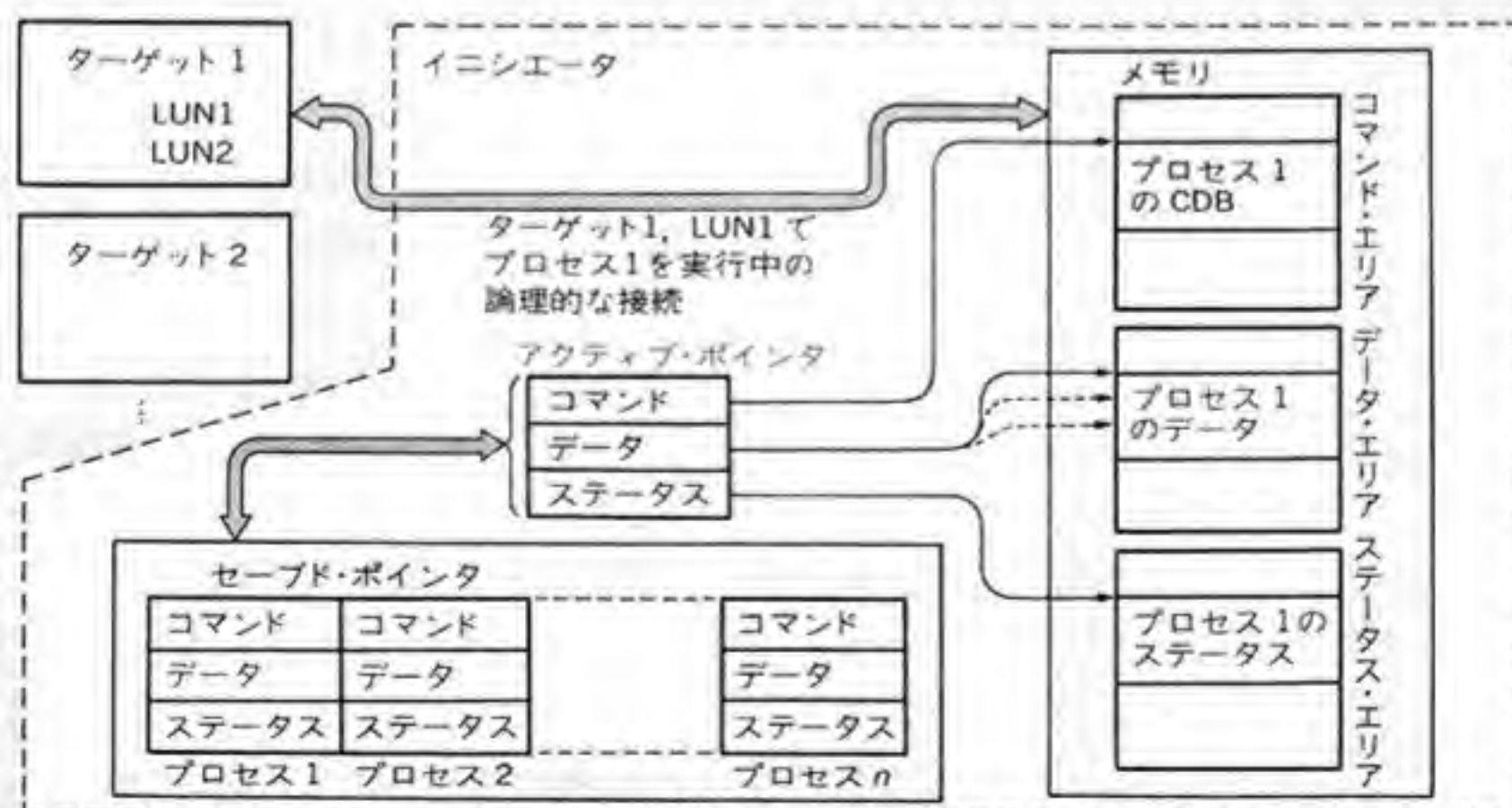
「SCSI ポインタ」という考えが必要です(実際にはイニシエータ内にある SCSI コントローラ・チップ内の各種レジスタやメモリがこの役目をするが、SCSI 規約ではそこまで立ち入れないので、「考え方」のみ示している)。このポインタは①コマンド、②データ、③ステータスの三つのポインタ・セットで構成され、イニシエータのメモリ上にあるコマンド、データ、ステータスの入っている、または入るべきアドレスを示します。ポインタ・セットは I/O プロセスごとに用意され、すべてセーブされています(セーブド・ポインタ)。ある I/O プロセスが実行されるとき(アクティブになったとき)そのプロセスのポインタ・セットは、1セットのみ用意されているアクティブ・ポインタに移され、コマンドやデータ、ステータスの転送制御などに使われます。セーブド・ポインタ内のコマンド・ポインタは、つねにその I/O プロセスの CDB の先頭アドレスを示しています。セーブド・ステータス・ポインタもまたそのプロセスのステータス・エリアの先頭を示していますが、セーブド・データ・ポインタのみは値を変化させることができます。アクティブ・データ・ポインタは転送制御に使われるので、(データの)転送時、値はどんどん更新されていきます。そのとき、ターゲットから Save Data Pointer メッセージがくると、その時点のアクティブ・データ・ポインタの内容がセーブド・データ・ポインタに移されます。これにより、データ転送途中であるプロセスを中断(実際は、ディスクコネク)した場合でも、そのプロセスを再開(実際は、

3

SCSI ポインタ

SCSI 上の I/O プロセスを制御・管理するためには、

図2 SCSI ポインタの概念図



リコネクト)したときセーブされていた値がアクティブ・ポインタに移るので、前に中断されたところから転送を再開できるわけです。イニシエータは、ある I/O プロセスが(初期でも中途でも)アクティブになったとき、必ずセーブド・ポインタ・セットをアクティブ・ポインタに移し、そのプロセスの制御を行います。

ターゲットが、イニシエータ内にあるポインタを操作する方法は、前述の Save Data Pointer メッセージのほか、Restore Pointer メッセージと、Modify Data Pointer メッセージがあります。Restore Pointer メッセージは、その時点でのその I/O プロセスのセーブド・ポインタ・セットをすべてのアクティブ・ポインタに移します。これにより、コマンド、データ、ステータスの再転送(リトライ)が可能となります。

Modify Data Pointer メッセージは、その時点のアクティブ・データ・ポインタの内容を変えるメッセージで、たとえば、ターゲットが何バイトかのライト・データを受け取ったけれども、どこかに失ってしまったような場合、その分、データ・ポインタをもとにもどして、そのデータをイニシエータに再送させることができます。

4

メッセージ・システムの詳細

イニシエータとターゲットには、実際の“仕事”であるコマンドやデータをやりとりする以外にこれらの“仕事”をうまく行うための“管理”が必要です。この



ネクサス(NEXUS)

SCSI 規格書には「ネクサス」という言葉がよく出てきます。これは「論理的なかわり(接続)」を意味し、初期接続(セレクション)により確立され、I/O プロセスの終了をもって完了します。基本はイニシエータ(I)⇄ターゲット(T)間の I_T ネクサスで、ロジカル・ユニット(L)を指定したり、キュー・タグ(Q)を指定することにより I_T_L、I_T_L_Q ネクサスとなると、範囲がより狭まった論理接続ということになります。

“管理”を行うのがメッセージ・システムの役割です。具体的には、イニシエータ⇄ターゲット⇄ロジカル・ユニットの論理的な接続を確立(Identify メッセージ)したり、データ転送方法の取り決めを行ったり、また通信エラー発生時の通報やコマンド・キューイングの管理なども行います。

メッセージはイニシエータ→ターゲット、ターゲット→イニシエータ双方向に有効ですが、メッセージの種類により一方専用メッセージと、方向によって若干取り扱いが異なってくるメッセージもあります。また、必須の(必ずサポートしなければならない)ものと、オプションのものがあって、自分のサポートしていない(またはできない)メッセージが相手から送られた場合、Message Reject メッセージを送り、それを拒否します(したがって、Message Reject メッセージは必須)。

1 章およびアテンション・コンディションの項でも説明しましたが、アクティブな I/O プロセス中、イニシエータは ATN 信号をアサートすることによりターゲットに対しメッセージ・アウト・フェーズを作る要求を行い、メッセージを送ることが可能となります。一つのメッセージ・フェーズ中に複数のメッセージを送ることは可能ですが、複数バイトのメッセージを複数のメッセージ・フェーズに分けて(またがって)送ることはできません。イニシエータからのメッセージの送出(メッセージ・アウト)で複数のメッセージを同一フェーズ中に送る場合、ATN をアサートし続けるわけですが、ATN 解除条件が“YES”となっているメッセージの場合は、この複数メッセージは送出できません(1 章の表 4、5 参照)。

セレクション・フェーズに続くイニシエータからの最初のメッセージは、Identify、Abort、Bus Device Reset のいずれかでなければなりません。これら以外のメッセージが送られた場合、ターゲットは“エラー”として扱い、バス・フリー・フェーズに入ってしまいます(Unexpected Disconnect)。

最初のメッセージが Identify である場合、これによりイニシエータ(I)⇄ターゲット(T)⇄ロジカル・ユニット(L)という論理的な接続(I_T_L ネクサス)が確立されます。また、リセレクション後のターゲットからのメッセージも Identify で、これにより I_T_L ネクサスは再確立されます。1 回の“接続”には一つのロジカル・ユニットしか指定できません。

さて、それぞれのメッセージごとにその意味と使用方を説明しましょう。コマンド・キューイング用以外のメッセージを、以下、アルファベット順にとりあげます。

■ Abort

このメッセージはイニシエータ→ターゲットの一方向メッセージです。現在実行中の I/O プロセスのクリアおよび I_T_L ネクサスが確立されている場合、その I_T_L ネクサス下にあるキューイングされているすべての I/O プロセスをクリアします。このメッセージを受け取ったターゲットは、バス・フリー・フェーズに移行しなければなりません。他の I_T_L ネクサスに属するデータ、ステータス、キューの中にある I/O プロセスは影響を受けません。I_T ネクサスのみしか確立されていない場合(セクション直後で Identify メッセージが送られる前)にこのメッセージが来たら、ターゲットはただたんにバス・フリー・フェーズに入るだけでその I/O プロセス、キュー内の I/O プロセスは影響を受けません。すでに設定済みの条件(モード・セレクトのパラメータ、装置のリザベーション、ECA 状態など)は、Abort によって変わりません。

■ Bus Device Reset

このメッセージは、イニシエータによって、SCSI バス上の特定のターゲットをハード・リセットするために使われます。ターゲットはこのメッセージを受け取った場合、バスを手放し、内部的にはリセット(初期化)を行います。したがって、その後、すべてのイニシエータに対しユニット・アテンション状態を作ります。

■ Command Complete

このメッセージは、ターゲットからイニシエータに送られ、一つの I/O プロセスの実行が終了したことを意味します。ターゲットはこのメッセージ送出後(ATN が“真”になっていなければ)バス・フリー・フェーズに入ります。

■ Disconnect

ターゲットからイニシエータにこのメッセージが送られる場合、イニシエータに対しターゲットが「これからディスコネクトしてバス・フリーにしますよ」という一方的な通知です。しかし、このメッセージには



I/O プロセスの中途終了

特定のターゲットの I/O プロセスを、通常終了以前に強制的に中途終了(クリア)させるメッセージは、Bus Device Reset, Clear Queue, Abort の三つです。それぞれのメッセージでクリアされる I/O プロセスは以下のとおりです。

- (1) Bus Device Reset: すべてのイニシエータからすべてのロジカル・ユニットに対する I/O プロセスがクリアされる
- (2) Clear Queue: すべてのイニシエータから特定のロジカル・ユニットに対する I/O プロセスがクリアされる
- (3) Abort: 特定のイニシエータから特定のロジカル・ユニットに対する I/O プロセスがクリアされる

「あとでまたリコネクトします」という約束も含まれています。

このメッセージがイニシエータに正常に受け取られ、ATN が“真”にされていなければ、ターゲットはバス・フリー・フェーズに入ります。データ転送途中でディスコネクトする場合で、あとで続きのデータ転送をしたい場合は、この Disconnect メッセージに先立ち Save Data Pointer メッセージを送る必要があります。

オプションですが、このメッセージはイニシエータからターゲットに送られることもあります。これは、イニシエータがターゲットに対し「ディスコネクトしてくれ!」という意味になります。ターゲットがこれをサポートしており、ディスコネクトが可能な場合、これに応えるかたちでメッセージ・イン・フェーズに移り、イニシエータに対し Disconnect メッセージを発行しなければなりません(できれば Save Data Pointer メッセージを出してから)。ターゲットはその後バス・フリーに入りますが、最短でもディスコネクト・ディレイ(200 μ s)の間は、アービトレーションに参加してはならない(リコネクトしようとしてはいけない)ことになっています。

■ Identify

このメッセージはイニシエータ/ターゲット両方より送ることのできるメッセージで、これにより I_T_L

図3 Identify メッセージのフォーマット



のネクサスが確立されます。Identify メッセージのフォーマットは図3のようになっています。

Identify ビット(ビット7)は"1"でなければなりません。これにより、このメッセージがIdentify メッセージであることを示します。DiscPriv(Disconnect Privilege)ビット(ビット6)は、イニシエータからの送出時のみ有効で、"1"であればターゲットがディスコネクトを行ってよいという許可の意味になります。"0"の場合、そのターゲットはそのイニシエータとのI/Oプロセス中ディスコネクトしてはいけません。LUNTAR ビット(ビット5)は、このIdentify メッセージがロジカル・ユニットの選択を行うものか、ターゲット・ルーチンを選択するものかを示します。"0"の場合、LUNTRN(ビット2～0)はロジカル・ユニット番号を示し、"1"であればLUNTRNはターゲット・ルーチンの番号を示します。ターゲット・ルーチンはターゲットの内部ルーチンで自己診断などを行うメンテナンス用のルーチンです。ターゲット・ルーチンをサポートしていないターゲットが、ターゲット・ルーチンの指定を受けた場合 Message Reject メッセージによりそのIdentify メッセージを拒否するか、Check Condition ステータスで応答します。

一つのI/Oプロセスの中では、ロジカル・ユニット番号を複数指定することはできません。イニシエータは、1接続(セレクションで始まりバス・フリーで終わる)中、Identify メッセージを複数回発行することが可能ですが、LUNTAR ビットおよびLUNTRN フィールドの値を変えてはいけません。したがって変えられるのは、DiscPriv ビットのみですが、これによりディスコネクトを許すか否かを途中で変更することが可能

です。

リコネクション時に、ターゲットからイニシエータに送られるIdentify メッセージは、イニシエータ内のポインタのリストAを暗黙のうちに指示しています。

■ Ignore Wide Residue

このメッセージは、ターゲットからイニシエータに送られる2バイト・メッセージです(図4)。

これはワイド・データ転送を行っている場合に、あるデータ・イン・フェーズでの最後のハンドシェイクの転送が規定のデータ幅を満たしておらず、無効データが入っているので無視してほしい旨の通知です。図4のようにIgnore フィールドで無視すべきデータを指示します。このメッセージはデータ・イン・フェーズの直後に送られなければなりません。

■ Initiate Recovery

このメッセージは、基本的にターゲットからイニシエータに送られるもので、ECA 状態をサポートしているターゲットがECA 状態に入ったことをイニシエータに通知するメッセージです。これは通常 Check Condition または Command Terminated ステータスの直後に発行されます。このECA 状態は通常、イニシエータからのRelease Recovery メッセージによって終わります。

また、非同期イベントが発生した場合、ターゲットは一時的にイニシエータとして動作しますが、「この一時的イニシエータ」の状態ではInitiate Recovery メッセージをSend コマンドに先立って送ることもできます(AEN: Asynchronous Event Notification; 非同期

図4 Ignore Wide Residue メッセージのフォーマットと Ignore バイトの意味

ビット バイト	7	6	5	4	3	2	1	0
0	メッセージ・コード="23h"							
1	Ignore							

Ignore バイト	32 ビット転送の場合の無効データ	16 ビット転送の場合の無効データ
00h	Reserved	Reserved
01h	DB(31~24)	DB(15~8)
02h	DB(31~16)	Reserved
03h	DB(31~8)	Reserved
04h~FFh	Reserved	Reserved



CA, ECA 状態

あるロジカル・ユニットが、あるイニシエータのコマンドを実行中、異常(エラー)が発生したとしましょう。その場合、そのイニシエータは、他のイニシエータに邪魔されずにエラー・リカバリができるように、エラーが発生したロジカル・ユニットの属しているターゲットをある程度独占的に使えるようになります。

まず CA(Contingent Allegiance: 臨時の忠誠)状態について説明しましょう。あるロジカル・ユニットがコマンドの実行中、エラーを検出した場合、そのロジカル・ユニットが属するターゲットは、そのコマンドを発行したイニシエータに Check Condition ステータスを送り、エラーの発生を知らせます(この時点で CA 状態となる)。このターゲットはそのイニシエータに対して、エラー内容を知らせるためセンス・データを準備し、同じイニシエータからつぎにそのロジカル・ユニットをアクセスされるまでこのセンス・データを保持していなければなりません。CA 状態の発生と終了は下の表のとおりです。

ターゲットは CA 状態中、そのセンス・データを維持しなければなりません。もし、他のイニシエータか

ら CA 状態を引き起こしたロジカル・ユニットに対するアクセスがあり、それを受け付けられセンス・データを維持できない場合は、そのアクセスを受け付けず、Busy ステータスを返すことができます。

CA 状態は“暗黙の了解”的エラー・リカバリ状態ですが、ECA(Extended Contingent Allegiance)状態はきわめて明示的です。まずイニシエータが ECA(Extended Contingent Allegiance)状態を使いたい場合、ターゲットに対し Mode Select コマンドによりその設定を行っておきます。するとロジカル・ユニットに異常が発生した場合、その異常内容によりターゲットが ECA 状態が必要と判断すると、Check Condition ステータスとともに Initiate Recovery メッセージを発行し ECA 状態に入ります。ECA 状態に入ったロジカル・ユニットは、他のイニシエータからのアクセスに対しすべて Busy ステータスで応答し、すなわちエラー・リカバリのために一つのイニシエータが排他的に使えるわけです。この状態はイニシエータが Release Recovery メッセージを発行することにより終了します。ECA 機能はオプションです。

CA の発生(始まり)	CA の終了	
Check Condition: ステータスの発行 または Command Terminated: ステータスの発行 オプション Unexpected Disconnect: (期待しないディスコネクト)	ハード・リセット	
	メ ッ セ ー ジ	そのイニシエータから、そのロジカル・ユニットに対する Abort
		すべてのイニシエータから、そのロジカル・ユニットに対する Clear Queue
		すべてのイニシエータから、そのターゲットに対する Bus Device Reset
	そのイニシエータから、そのロジカル・ユニットに対するコマンド	Request Sense 以外のコマンドでは即終了 Request Sense コマンドでは、センス・データの送付をもって終了

イベントの通知)。この場合も Release Recovery メッセージで ECA 状態を終了します。ターゲットがこの Initiate Recovery を送ったのに、イニシエータが Message Reject メッセージで応答(拒否)した場合は ECA 状態になりませんが、この“拒否”により ECA モードの設定は変わりません。つまり“拒否”されても、ターゲットがこの「Initiate Recovery メッセージを送る」という設定(モード・セレクト・コマンドで設定する)は変化しません。

■ Initiator Detected Error

このメッセージはイニシエータが、なんらかのエラーを検出したことをターゲットに知らせるメッセージで、ターゲットによるリトライ(前のフェーズのやり直しなど)を許す意味も含まれています。エラーの原因は、イニシエータ内部にあることもあり、また、SCSI バス上に関連していることもあります。イニシエータ内のエラーですからポインタ内の値も保証されているわけではありませんが、ターゲットとしては Restore Pointer メッセージ、またはディスコネクト/リコネクトを行うことによりポインタ値を一つ前に戻しリトラ

イすることもできます。

■ Linked Command Complete

このメッセージはターゲットからイニシエータに送られます。リンクされたいくつかのコマンドの一つが終了し、そのコマンドに対するステータスが正常に返送されたことを意味します。このメッセージにより、イニシエータはポインタをつぎのリンク・コマンドの先頭にセットしなければなりません。

■ Linked Command Complete (With Flag)

フラグ付きのリンクド・コマンドに対して使われる上記 Linked Command Complete と同じ意味のメッセージです。イニシエータ内ではフラグ付きコマンドの終了によりなんらかの割り込みに使ったりします。

■ Message Parity Error

このメッセージは、イニシエータからターゲットに送られ、イニシエータが最後に受け取ったメッセージ・バイトにパリティ・エラーがあったことを示します。



AEN(Asynchronous Event Notification ; 非同期イベントの通知)

ロジカル・ユニットあるいはターゲット側に、SCSI バスとは無関係(非同期)のできごと(イベント)が発生した場合、たとえばハード・ディスクのスピンドル・モータが停止したり、テープ装置のテープが交換されたり、またロジカル・ユニットの電源が ON/OFF されたときなど、通常はターゲット側からイニシエータに積極的に知らせることはせず、そのイベント後の最初のコマンドに対し Check Condition ステータスを送り、センス・データでその内容(ハードウェア・エラーやユニット・アテンションなど)を報告します。つまり、アクセスされないかぎり非同期イベントは報告しないのですが、これをターゲット側から積極的に通知するオプション機能が AEN です。AEN の手順はおおまかに以下のようになっています。

- (1) 周辺装置ターゲットは、パワー・オンされたあと、SCSI 上につながれている自分以外のすべての ID の LUN 0 に Inquiry コマンドを発行し、どの装置が AEN の受け付けができるプロセッサ装置(すな

わち通常イニシエータとして働く装置)かを調べます(このとき、このターゲットはイニシエータとして動作しています)。

- (2) このターゲットに属するロジカル・ユニットに何か非同期イベントが起きた場合、(1)で調べた結果より通知を受け取ってくれるプロセッサ装置に Send コマンドを使ってこのできごとを通知します。通知の内容は Request Sense コマンドのセンス・データと同じ形式です。

- (3) この通知(AEN)に対し、各プロセッサ装置は対処方法を考えます。

非同期イベントが ECA 状態を引き起こすこともあります(これはターゲットの判断による)。その場合は Send コマンドの前に Initiate Recovery メッセージを送り ECA 状態に入ります。

このように AEN の手順を行うためには両方の装置がイニシエータにもターゲットにもなれなければなりません。

どのメッセージ・バイトにエラーがあったかをターゲットに正しく認識させるため、イニシエータはパリティ・エラーを検出したバイトの転送の ACK を“偽”にする前に ATN を“真”にしなければなりません。ターゲットはこの ATN に応える形でメッセージ・アウトに移り、イニシエータはこの Message Parity Error を送ることができるわけです。このメッセージに対しターゲットがメッセージ・イン・フェーズに移った場合、ターゲットは必ずエラーの検出されたメッセージを全バイト、再転送しなければなりません。

上記タイミング以外のときにこのメッセージがターゲットに送られた場合、ターゲットは致命的なエラーと認識し、バス・フリーに移行します(Unexpected Disconnect)。

■ Message Reject

このメッセージはイニシエータからでもターゲットからでも送ることができ、最後に送られたメッセージまたはメッセージ・バイトが、不適切か、あるいはサポートされていないことを意味します。

イニシエータがこのメッセージを送る場合、どのメッセージをリジェクトしたかをターゲットに正しく認識させるために、イニシエータはリジェクトすべきメッセージ・バイトの転送の ACK を“偽”にする前に ATN を“真”にし、メッセージ・アウト・フェーズの要求をしなければなりません(Message Parity Error と同じタイミング)。このタイミング以外のとき、このメッセージ(Message Reject)がきた場合、ターゲットはまた Message Reject メッセージを出して、そのイニシエータからきた Message Reject を拒否します。

ターゲットがこのメッセージを送る場合、リジェクトしたいメッセージ・バイトを受け取った直後、メッセージ・イン・フェーズに移り、この Message Reject



コマンド・リンク

あるイニシエータが一つのロジカル・ユニットに対し複数のコマンドを連続して発行したほうが都合がよい場合(たとえばハード・ディスクでサーチ→リードとかテープ装置でスペース→リードなど)コマンドをリンクしておくことができます。コマンド(CDB)の最後のバイト(コントロール・バイト)にリンク・ビットがあり、これを“1”としておくことにより、ターゲットはこのコマンド終了後、つぎのコマンドがあることを認識します。SCSI の規約では、リンクされたコマンド列は一つの I/O プロセスとして取り扱われますので、あるコマンドの終了後、バス・フリーに入る必要がなく、また L_T_L ネクサスの再確立(セレクションや Identify メッセージ)が必要ないので、イニシエータ側の処理を簡単にしたり、バスの使用効率を上げることができます。CDB 中のリンク・ビットとともにフラグ・ビットを“1”とした場合、コマンド・コンプリートのメッセージも異なったコードで示されるので、リンクされたコマンド列のうちいくつかをフラグ付きにし、ホスト側で割り込みを発生させたりもできます。

メッセージを送らなければなりません。これによりイニシエータは、どのメッセージ・バイトがリジェクトされたか正しく認識できるわけです。

ターゲットがこの Message Reject メッセージを送ったにもかかわらず、イニシエータの ATN が“真”になっている場合、ターゲットはメッセージ・アウト・フェーズに移りイニシエータのメッセージを受け取りますが、イニシエータからのメッセージは、(複数バイト・メッセージの場合)最初のバイトから再スタートして転送されなければなりません。

図5 Modify Data Pointer メッセージ

バイト	値	内容(意味)
0	01h	拡張メッセージ
1	05h	拡張メッセージ長
2	00h	Modify Data Pointer コード
3	X	アークギュメント(MSB)
4	X	アークギュメント
5	X	アークギュメント
6	X	アークギュメント(LSB)

■ Modify Data Pointer

このメッセージはポインタの項で述べたようにターゲットがデータを失ったような場合に威力を発揮します。図5のようになっている、ターゲットからイニシエータに送られ、データ・ポインタの変更を要求する7バイトのメッセージです。イニシエータは自分のアクティブ・ポインタにメッセージ内のアークギュメント・バイト(4バイト、2の補数形式)の値を加算しま

す。

■ No Operation

このメッセージは、イニシエータからターゲットに送られます。意味はありません。イニシエータが送るべきメッセージがないのに、ターゲットがメッセージ・アウト・フェーズに移ったときに使います。たとえば、イニシエータが ATN を立てたのに、ターゲットがそのときにはメッセージ・アウト・フェーズにしてくれず、しばらくしてメッセージ・アウトになったとき、送ろうとしていたメッセージが無意味になってしまった場合などに使います。

■ Release Recovery

このメッセージはイニシエータからターゲットに送られ、ターゲットが Initiate Recovery によって確立した ECA 状態を終了させるために使われます。このメッセージは Identify メッセージの直後に同じメッセージ・アウト・フェーズ中に送られなければなりません。

これを受け取ったターゲットは、ECA 状態を解除し、すぐにバス・フリー・フェーズに移行しなければなりません。ECA 状態でなかったのに、ECA をサポートしているターゲットに対しこのメッセージが送られた場合、このターゲットはとくにメッセージをリジェクトせず、すぐバス・フリー・フェーズに移行しなければなりません。

■ Restore Pointer

このメッセージはターゲットからイニシエータに送られ、現在の I/O プロセスのもっとも新しいセーブド・ポインタの全内容(データ、コマンド、ステータスの三つのポインタ)をアクティブ・ポインタにコピーさせます。結果として、コマンドおよびステータスのポ

インタは、そのときの両エリアの先頭番地を示さなければなりません。データ・ポインタは、もしこのネクサスに Save Data Pointer メッセージが一度でも発生していれば「最後にセーブされた値」を、そうでなければ「データ・エリアの先頭」を示します。

■ Save Data Pointer

このメッセージはターゲットからイニシエータに送られ、その I/O プロセスのアクティブ・データ・ポインタの現在値をセーブド・データ・ポインタ内にコピーさせます。

● Synchronous Data Transfer Request (SDTR) (図 6)

このメッセージは、イニシエータ/ターゲット間で交換され、データの同期転送の設定に使われます。最初はどちらから、という決まりはないので、イニシエータでもターゲットでも必要と思ったほうが思ったときに(若干制限はありますが)発行できます。規格では「以前設定されたデータ転送に関する同意が無効となった可能性があればいつでもこのメッセージの交換を行わなければならない」となっています。

データ転送に関する同意が無効となる可能性のある状態は、たとえばハード・リセット後、Bus Device Reset メッセージ後やパワー・オン/オフ後などです。そのほかのときでも、この SDTR の交換が有益であると考えた場合は、交渉(ネゴ)ができます。同期転送をサポートしているときは、いつなんどき相手から SDTR メッセージが来ても、それを Message Reject によって拒否してはいけません。

ただし、極端な性能の向上が期待できないかぎり、セレクションのたびにネゴすることは推奨されていません。

また、昔のホスト・アダプタで SDTR メッセージを受け付けないものもあるため、ターゲットはパワー・オン後(この規格では SDTR の交換をしなければならないことになっているが)、自分から SDTR を発行しなくてもよいことになっています。さらに、このようなターゲットが(同期モードを設定されていたにもかかわらず)パワー・オフ/オンされた場合、イニシエータ/ターゲット間で「思い違い」ができてしまうので、Request Sense あるいは Inquiry (通常は異常発生時やパワー・オン直後にしか使われない)コマンドごとに、

図 6 Synchronous Data Transfer Request

バイト	値	内容・意味
0	01h	拡張メッセージ
1	03h	拡張メッセージ長(3 バイト)
2	01h	SDTR コード
3	m	転送周期(m×4 n 秒)
4	X	REQ/ACK オフセット

イニシエータは SDTR によるネゴを行ってもよいことになっています。

SDTR メッセージの交換の結果として、データ転送時の許容される(最小の)転送周期と(最大の)REQ/ACK オフセット値が設定されます。この同意による転送はデータ・フェースでのみ有効です。

転送周期は、REQ および ACK の立ち上がりからつぎの立ち上がりまでの最小時間です。また、REQ/ACK オフセット値は、対応する ACK による応答が返る前に先行できる REQ パルスの数です。このオフセット値が“0”である場合は「非同期転送」を意味し、“FF”は「制限なし」を意味します。

ネゴの方法はつぎのとおりです。

- (1) ある SCSI デバイス(言い出しっべ)は、自分で取り扱える数値(通常は最高スピード、最大オフセット)を入れた SDTR メッセージを相手に送る。
- (2)-1 相手側は(1)の数値が自分でもハンドリングできる範囲内であれば、同じ数値の SDTR メッセージを送る(ネゴ終了)。
- (2)-2 もし(1)の数値が自分でハンドリングできる範囲を超えていたら、ハンドリングできる数値を入れた SDTR メッセージを返す(ネゴ終了)。

このネゴの結果は、最小転送周期と最大オフセットを二つのデバイス間で同意したもので、実際の転送はこの範囲外でなければ、どんな値をとってもかまいません。ネゴが正しく終了しても、オフセット値が“0”は非同期転送での同意を意味します。また、SDTR メッセージが Message Reject メッセージによって拒否された場合も、非同期転送での同意を意味します。

ネゴ中になんらかの異常、たとえば Unexpected Disconnect、リトライでも回復できなかったメッセージ・パリティ・エラーなどが起きた場合も、すべての非同期転送モードで終了したことになります。このネゴの結果は、リセットまたはつぎのネゴまで両デバイス間で有効です。すべてのデバイスで、デフォルトは非同期転送モードです。

■ Terminate I/O Process

このメッセージはイニシエータからターゲットに送られ、ターゲットに対し現在の I/O プロセスの“中止”を要望するものですが、「媒体をこわさずに」の条件付きです。「媒体をこわさずに」とは、たとえばハード・ディスクのセクタの途中で書き込みを中止しない

図7 Wide Data Transfer Request

バイト	値	内 容
0	01h	拡張メッセージ
1	02h	拡張メッセージ長
2	03h	WDTR コード
3	m	転送幅(2 ^m バイト)

m=0 → 1 バイト(8 ビット)
m=1 → 2 バイト(16 ビット)
m=2 → 4 バイト(32 ビット)

(ECC まできちんと書く)といった意味です。これを受けたターゲットは、“できるだけ早く”その I/O プロセスを中止し、Command Terminated ステータスを返さなければなりません。そのときのセンス・キーは No Sense で、アディショナル・センス・コード/クオリファイアは I/O Process Terminated となります。このメッセージによる I/O プロセスの中止は、他の I/O プロセスには影響してはなりません。同じネクサスを使うキューの中の、続く I/O プロセスには若干影響する可能性もあります。

中止を要望された I/O プロセスの実行中にターゲットがエラーを検出した場合は、この Terminate I/O Process メッセージは無視され、イニシエータには通常どおりエラーが報告されます。また、I/O プロセスが中止できずに、完全に終了してしまった(「時、すでに遅し」)場合も、このメッセージは無視され、ターゲットは通常終了を報告します。

もし、このメッセージがコマンドを受け取る前にきた場合は、ターゲットは Command Terminated ステータスでそのプロセスを終了し、センス・データも No Sense+I/O Process Terminated となります。

対象となる I/O プロセスが、まだコマンド・キュー内にあり実行されていない場合には、ターゲットは即座にそのプロセスを終了してもよいし、とりあえずディスクコネクトし、そのプロセスがキューの先頭にきてから終了してもかまいません。

ターゲットがこのメッセージをサポートしていないか、その I/O プロセスを中止できない場合、Message Reject メッセージにより拒否し、その I/O プロセスを普通に実行しなければなりません。

通常、このメッセージは、イニシエータにその I/O プロセスよりも優先度の高いコマンドが発生してしまい、今、ターゲットがやっていることを早めに中止さ

せたいときに使われます。中止させたコマンド(プロセス)をあとでもう一度やらせるかどうかはイニシエータの勝手です。

■ Wide Data Transfer Request (WDTR)

(図7)

このメッセージはワイド転送を設定するものですが、説明はSDTRとほとんど同じです。ただし、SDTRでは「非同期転送」であったものをWDTRでは8ビット転送と読みかえます。さらに、ワイド転送はあまり普及していないようです。

一つだけ注意しておきたいことは、このWDTRのネゴはSDTRの解除条件になっていることで、すなわち、SDTRで同期転送を設定したあと、このWDTRでワイド転送を設定すると、SDTRで設定した同期転送は解除され非同期転送になっています。したがって同期転送(およびFAST転送)とワイド転送を併用したい場合は、WDTRによるネゴのあと、SDTRによるネゴを行わなければなりません。

■ コマンド・キューイング関係のメッセージ

コマンド・キューイング用メッセージは全部で5種類ですが、そのうち三つはイニシエータがコマンドにタグ(荷札、番号)をつけるためのキュー・タグ・メッセージという2バイトのメッセージで、他の二つはAbort TagおよびClear Queueという制御用1バイト・メッセージです。

● キュー・タグ・メッセージ(図8)

キュー・タグ・メッセージは2バイト長で1バイト(バイト0)は、メッセージ・コード、2バイト目(バイト1)はコマンドごとの番号、すなわちタグが入ります。タグは0~255までの数値です。

以下、アルファベット順にキュー・タグ・メッセージについて説明します。

図8 キュー・タグ・メッセージ

バイト	7	6	5	4	3	2	1	0
0	メッセージ・コード(20hまたは21hまたは22h)							
1	キュー・タグ(00h~FFh)							

(1) Head of Queue Tag(21h)

このメッセージ付きのコマンドは、キューの先頭におかれ、現在実行中のコマンドが終了次第実行されます。優先度の高いコマンドに対して、イニシエータが付けます。

(2) Ordered Queue Tag

このメッセージ付きのコマンドは「到着順」に実行されます。すなわち、そのときキューに入っているコマンドの最後に置かれ、先着コマンドすべてを実行後に実行されます。また、このコマンドより後に到着したコマンドは、Head of Queue Tagメッセージ付きのコマンドを除き、このコマンドより後で実行されます。このメッセージは、イニシエータ処理の都合上、実行順序を変えられたくない場合(たとえば、テープ装置でリワインド後にリードしたい場合など)に使います。

(3) Simple Queue Tag

このメッセージ付きでイニシエータから発行されたコマンドは、ターゲットが実行順序を変更することができます。この場合、ターゲットは効率のよい順序でこのコマンドを実行できます(たとえばハード・ディスクで、シーク方向順に実行する)。

また、このメッセージはターゲットも使います。ターゲットは、タグ付きのコマンドをリコネクトする際、どのコマンドのリコネクションかをイニシエータに知らせるため、このメッセージによりタグ番号を通知します。

● コマンド・キューイング制御用メッセージ

(1) Abort Tag

このメッセージはイニシエータから送られ、そのイニシエータが発行した、ある一つのタグ付きコマンドをクリアします。そのコマンドがSCSI上で実行中の場合はこのメッセージのみでクリアできますが、ディスクコネクト中の場合は、セクション→Identifyメッセージによるロジカル・ユニットの指定→Simple Queue Tagメッセージによるコマンドの指定のあと、このメッセージを送らなければなりません。ターゲットはこのメッセージを正しく受け取るとディスクコネクトします。

(2) Clear Queue

このメッセージは、イニシエータが特定のロジカル・ユニットのキュー内および実行中のコマンドをす

べてクリアします。他のイニシエータが発行したコマンドもクリアされますので、それらのイニシエータに対してはユニット・アテンション・コンディションとなります。ターゲットはこのメッセージを正しく受け取ると、バス・フリー・フェーズに移行します。

5

ステータス

第1章ではステータスに関して軽くしかふれていませんので、ここで詳しく説明しておきます(1章の図13参照)。

ステータスは、一つ一つのコマンドの終了ごとにターゲットからイニシエータに送られる1バイトのコードです。当然、ステータス・フェーズ中に送られますが、つぎの場合のコマンド終了では、ステータスは発生しません。

▷ Abort, Bus Device Reset, Clear Queue メッセージによるコマンドの終了

▷ ハード・リセット

▷ Unexpected Disconnect

上記以外の場合、コマンドの終了時には必ずステータスが送られなければなりませんが、コマンドが送られてくる前にステータスを発生させることもあります。

それぞれのステータスの意味は以下のとおりです。

▷ Good: ターゲットがコマンドを正常に終了したことを示す。

▷ Check Condition: CA 状態が発生したことを示す。つまり、コマンド実行中に何か異常(エラー)が発生し、イニシエータによるリカバリが必要であることを意味する。

▷ Condition Met: Search Data 系のコマンドを実行した結果、条件を満たしたデータ検索に成功したことを示す。ターゲットはその探し出したアドレスを含んだセンス・データを用意する。

▷ Busy: これは、ターゲットがビジー(忙しい)ことを意味する。ターゲットはイニシエータからのコマンドを(他のイニシエータにリザーブされていないのに)受け付けられない場合、いつでもこのステータスを送ることができる。この場合、そのイニシエータは、あとでもう一度コマンドを送ることが推奨される。

▷ Intermediate: 一連のリンクされたコマンド群の中の、最終コマンド以外の(一つの)コマンドが正常終了したことを示す。

▷ Intermediate-Condition Met: Intermediate ステータスと Condition Met ステータスの複合された意味をもつ。リンクされた一連のコマンド実行の途中でこのステータスと上記 Intermediate 以外のステータスが発行された場合は、その I/O プロセスが中止されたことを意味する。

▷ Reservation Conflict: イニシエータがアクセスしようとしたロジカル・ユニットが、他のイニシエータにリザーブされていることを意味する。

▷ Command Terminated: このステータスは、Terminate I/O Process メッセージの結果として、ターゲットが I/O プロセスを中止した場合に送られる。同時に CA 状態の発生も意味する。

▷ Queue Full: このステータスは、コマンド・キューがいっぱいで、これ以上タグ付きコマンドを受け取ってもキューイングできないことを示す。

6

コマンド・オーバビュー

SCSI の基本的な考えは、「同一の環境で、機能・能力の異なるさまざまな周辺機器を幅広く扱える」ことにあります。今まで出てきた物理的接続、フェーズ、論理的接続の確立などは比較的簡単に、しかも理解しやすい形で規格統一ができていました。コマンドに関しても、プロトコル(やりとりの方法・約束事)は基本的なこととしてシンプルに統一化されていますが、コマンドはそう簡単にはいきません。ほんとうに基本的なコマンドと、SCSI インターフェース上の事柄をつかさどるコマンド以外は、コマンドそのものが装置の動作・機能・性能に直接関わってくるため、「統一」は非常にむずかしいといえるでしょう。まず第一に、装置の種類(デバイス・タイプ)によってアクセスする媒体が違っているので、別々のコマンド群ができます。第二に、ユーザのニーズの多様化とメーカ側の差別化があげられます。ユーザが「こんな機能があったらいいな」と思ったり、メーカが「こんな機能をつけたらあそこの装置より売れるだろう」といった思いをすべて「統一規格」にすることはほとんど不可能でしょう。しかし、

SCSI の規格化を行っている ANSI のワーキング・グループは、各分野の専門家を擁し、この不可能と思われる作業を行って、満足のいく結果を出しているといえます。このため、規約の中でもコマンドに関する記述はもっとも大きな部分を占めていて、なおかつオプションとして位置づけられているパラメータが大変多くなっています。昔は、メーカーによる“差別化”を、ベンダ・ユニークとかベンダ・スペシフィックといった形で実現していましたが、これはユーザにとって、決して便利なものではありませんでした。現在の規約

のもとでは、ベンダ・ユニークの部分で極力減らし、オプション機能による“差別化”を促しているようです。これにより、利用者側は機能の違いを比較しやすく、またメーカー側も新機能の採用・追加がやりやすくなったといえるでしょう。

このようなことからわかるように、ある SCSI 装置を使用する際は、SCSI の規格書を見て、ではなくその装置の仕様書やマニュアルを見てインストールすることをお奨めします。装置によってサポートしている機能レベルが異なるためです。理想的には、SCSI 規格

図9 6バイト・コマンドの CDB(代表的なもの)

ビット バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード							
1	ロジカル・ユニット番号 (LUN)				(MSB)			
2	ロジカル・ブロック・アドレス (必要に応じて)							
3								
4	(MSB) 転送長または パラメータ・リスト長または アロケーション長 (必要に応じて)							
	(LSB)							
5	コントロール・フィールド							

図10 10バイト・コマンドの CDB(代表的なもの)

ビット バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード							
1	ロジカル・ユニット番号 (LUN)				Reserved			
2	(MSB) ロジカル・ブロック・アドレス (必要に応じて) <							

図11 12バイト・コマンドの CDB(代表的なもの)

ビット バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード							
1	ロジカル・ユニット番号 (LUN)				Reserved			
2	(MSB) ロジカル・ブロック・アドレス (必要に応じて)							
3								
4								
5								
6	(LSB)							
7	(MSB) 転送長または パラメータ・リスト長または アロケーション長 (必要に応じて)							
8								
9								
10								
11	(LSB)							
12	Reserved							
13	コントロール・フィールド							

図12 オペレーション・コード

ビット	7	6	5	4	3	2	1	0
	グループ・コード				コマンド・コード			

図13 コントロール・フィールド

ビット	7	6	5	4	3	2	1	0
	ベンダ定義			Reserved			フラグ	リンク

をすべて完璧に盛り込んだデバイス・ドライバがあり、どんな装置を接続されても、それなりに使えればよいわけですが、ソフト的・ハード的なオーバーヘッドを考えるとあまり現実的ではないと思います。近い将来はそんな形になるかもしれませんが……。

さて、ここではコマンド・プロトコルの基本である CDB を第一に、そして全デバイス・タイプ共通コマンドとその他の主要コマンドについて、あまりしつこくならないように説明していくことにします。

■ CDB (Command Descriptor Block)

図 9, 10, 11 に 6 バイト・コマンド、10 バイト・コマンド、12 バイト・コマンドの CDB のフォーマットを示します。CDB はコマンドの記述型式で、イニシエータは必ずこの形でターゲットにコマンドを送ります。CDB の 1 バイト目(バイト 0)はオペレーション・コードと呼ばれる命令(リードとかライト)そのものです。最後のバイトはコントロール・フィールドと呼ばれ、制御コードが入ります。また、コマンドによっては CDB のあとに色々なパラメータが続くものがありますが、パラメータはデータ・アウト・フェーズで送られます。

それでは CDB 内の各フィールドについて説明しましょう。

● オペレーション・コード

オペレーション・コード・フィールドは、図 12 のようにグループ・コードとコマンド・コードに分けられます。したがって一つのグループには 32 種類のコマンドを割り当てることができます。グループは以下のように決められています。

- ・グループ 0: 6 バイト・コマンド
- ・グループ 1 および 2: 10 バイト・コマンド
- ・グループ 3 および 4: Reserved
- ・グループ 5: 12 バイト・コマンド
- ・グループ 6 および 7: ベンダによる定義

● ロジカル・ユニット番号(LUN)

SCSI-2 では、LUN は Identify メッセージで指定され、I_T_L ネクサスを確定するため、この CDB 内の LUN は意味をもちません。SCSI-1 しかサポートしていない装置との互換性を考えここに残してありますが、SCSI-2 ではこのフィールドを“0”とすることを推奨し

ています。

● ロジカル・ブロック・アドレス

ロジカル・ユニットまたは 1 装置を論理的にいくつかに分割した場合の一つの部分(パーティション)のブロックのアドレスは 0 から始まり、連続した番号(0, 1, 2…)で示されます。これがロジカル・ブロック・アドレスです。1 ブロックの大きさは通常 256 バイト、512 バイトあるいは 1024 バイト程度が多いようです。パーティションは、使い方にもよりますが、数 10~数 100 M バイトで切られることが多いようです。

6 バイト・コマンドでは 21 ビット、10/12 バイト・コマンドでは 32 ビットでこのアドレス指定ができます。

具体的な例をあげると、リード・コマンドの場合「どこから読み出せ」の“どこから”にあたります。

● 転送長

転送長は、転送されるべきデータ量を示します。通常、ブロック数で取り扱われることが多いのですが、コマンドによっては転送されるべきバイト数の場合もあります。転送長フィールドを 1 バイトしかもっていないコマンドでは、転送長“0”は 256 を意味します。転送長フィールドを複数バイトもっているコマンドでは転送長“0”はデータ転送なしを意味し、1 以上の転送長はそのままの数値を意味します。

具体的な例をあげると、リード・コマンドの場合「どれだけ読み出せ」の“どれだけ”にあたります。

● パラメータ・リスト長

このフィールドがパラメータ・リスト長となっている場合は、そのコマンド実行のためイニシエータからターゲットへ、データ・アウト・フェーズ中に転送されるデータ(パラメータ)のバイト数を表します。パラメータの例としては、モード・パラメータ、ログ・パラメータなどがあげられます。“0”の場合はデータ転送はありません。

● アロケーション長

アロケーション長は、ある種のコマンドでターゲットからイニシエータに情報(データ)が返ってくる場合、イニシエータが用意しているデータ受け入れ場所の大きさをバイト数で表したものです。ターゲットはアロ

ケーション長をこえてデータを転送してはいけません。この場合のデータ(情報)は、センス・データ、モード・データ、ログ・データ、インクワイアリ(Inquiry)・データなどです。“0”の場合はデータ転送はありません。

● コントロール・フィールド

コントロール・フィールドは図13のようになっています。

▷ フラグ・ビット

フラグ・ビットは、リンク・ビットが“1”になっているコマンドを実行し、エラーなしで終了した場合にターゲットからどのメッセージを送るかを指定するビットです。このビットが“1”の場合は Linked Command Complete (With Flag)、“0”の場合、ただの Linked Command Complete メッセージになります。リンク・ビットが“0”のときはこのビットは必ず“0”でなくてはなりません。

▷ リンク・ビット

このビットが“1”になっている場合は、このコマンドの実行終了後も同じ I/O プロセスを引き続き継続するようイニシエータがターゲットに要求しているこ

とを意味します。リンク・ビットが“1”になっているコマンドを正常終了した場合、ターゲットは Intermediate または Intermediate-Condition Met ステータスを送り、その後、フラグ・ビットで指定されたメッセージを送ってからさらにコマンド・フェーズに移り、つぎのコマンドを受け取らなければなりません。

■ 全デバイス・タイプ共通コマンド

全デバイス・タイプ共通コマンドは表2のとおりです。このうち Inquiry, Request Sense, Send Diagnostic, Test Unit Ready の四つのコマンドは、すべてのターゲットに対しサポートが義務づけられています。

それでは個々のコマンドの説明をしていきましょう。Request Sense は発生したエラーを知るため、Inquiry は装置そのものを知るため、また Mode Select/Mode Sense は装置の色々な設定を行ったり、設定状態を知るために、それぞれ重要なコマンドですので、すこし詳しく説明します。

● Test Unit Ready(OP コード=00h)

ロジカル・ユニットがレディ(使用可能)かどうかを

表2 全デバイス・タイプ共通のコマンド・セット

区分	OP コード	コマンド名称	SCSI-2	CCS	SCSI-1
グループ 0	00h	Test Unit Ready	M	M	O
	03h	Request Sense	M	M	M
	12h	Inquiry	M	M	E
	15h	Mode Select(6)	Z	O	O
	18h	Copy	O	O	O
	1Ah	Mode Sense(6)	Z	O	O
	1Ch	Receive Diagnostic Results	O	O	O
	1Dh	Send Diagnostic	M	M	O
グループ 1	39h	Compare	O	O	O
	3Ah	Copy And Verify	O	O	O
	3Bh	Write Buffer	O	O	Rsvd
	3Ch	Read Buffer	O	O	Rsvd
グループ 2	40h	Change Definition	O	Rsvd	Rsvd
	4Ch	Log Select	O	Rsvd	Rsvd
	4Dh	Log Sense	O	Rsvd	Rsvd
	55h	Mode Select(10)	Z	Rsvd	Rsvd
	5Ah	Mode Sense(10)	Z	Rsvd	Rsvd

M: サポート必須, E: 拡張仕様, O: オプション, Z: デバイス・タイプ依存, Rsvd: リザーブ

調べるコマンドです。レディ状態とは、あくまで“使用可能”な状態のことで、“応答可能”ではなく、ロジカル・ユニット本来の使い方ができるかということです。したがって、メディアを交換できる装置では、メディアが入っているかどうかのチェックにも使えます。レディ状態のときは Good ステータスが返ります。

● Request Sense(03h)

その名のとおりに、センス・データを要求するコマンドです。基本的には、エラー発生時(CA, ECA 状態発生時)にその内容を知るために使われます。SCSI-1 ではエラーの概要ともいえる 4 バイトのセンス・データ形式と、詳しい情報の含まれた拡張センス・データ(8 バイト以上)がありましたが、SCSI-2 では 4 バイト形式は削除され、最小でも 18 バイトの形式となりました(図 14)。各フィールドの意味を簡単に説明しておきます。

〈有効; Valid〉

バイト 3 ~ 6 のインフォメーション・フィールドの内容が SCSI-2 規格にのっとっているか、そうでないかを示します。“1”=SCSI-2 標準。

〈エラー・コード; Error Code〉

現在のところ 70h と 71h しか定義されていません。7Fh 以上はベンダ固有の定義が可能です。70h はカレント・エラー(現在エラー)と呼ばれ、Check Condition または Command Terminated ステータスを返してきた、そのコマンド実行中のエラーのセンス・データであることを示します。71h はデファード・エラー(据え置きエラー)と呼ばれ、(バス上では)すでに実行を終え、Good ステータスを返したコマンド(イミディエート・ビット付きのコマンドや、キャッシュ動作などにより Good ステータスの先送りがある)に対してのエラーのセンス・データであることを示します。

〈セグメント番号; Segment Number〉

コピー・コマンドやコンペア・コマンドなどのように 1 命令でも「セグメント」を使うことにより媒体上の複数の領域にアクセスできるコマンドの場合、どのセグメントでのエラーかを示します。

〈ファイルマーク; Filemark〉

シーケンシャル・アクセス・デバイスでは必ずサポートが必要となっています。ファイルマークまたはセットマークを検出した場合、このビットは“1”となります。

図14 センス・データ・フォーマット

ビット バイト	7	6	5	4	3	2	1	0	
0	有効	エラー・コード(70h または 71h)							
1	セグメント番号								
2	Filemark	EOM	ILI	Rsvd	センス・キー				
3	(MSB)	情報フィールド							(LSB)
6									
7	追加センス長								
8	(MSB)	コマンド固有情報							(LSB)
11									
12	追加センス・コード								
13	クオリファイア								
14	FRU コード								
15	SKSV	センス・キーによる定義							
17									
18	追加センス・バイト								
n									

〈EOM; End of Medium〉

これはシーケンシャル・アクセス・デバイスとプリンタ・デバイスでサポートが必須となっています。このビットが“1”のときの意味はひとくちでエンド・オブ・メディアですが、たとえばテープ装置でフォワード方向のとき最終記録領域にきたことを示したり、物理的なテープの終端に達したことや、リバース方向では先頭(BOT)まで達したことを示します。またエンド・オブ・パーティションでも使われます。プリンタでは紙切れを示します。

〈ILI; Incorrect Length Indicator〉

“1”の場合、要求された論理ブロック長が装置上のブロック長と一致していないことを意味します。

〈センス・キー; Sense Key/追加センス・コード; Additional Sense Code/追加センス・コード・クオリファイア: Additional Sense Code Qualifier (ASCQ)〉

この三つのコードでエラーの内容を詳しく知ること

ができます。センス・キーはエラーの概要を、ASC はその内容を詳しくしたもので、ASCQ は ASC をさらに掘り下げたものです (p.156 の付録にエラー・コードの一覧表があります)。

〈情報フィールド：Information〉

このフィールドはデバイス・タイプにより、またエラー発生時に実行していたコマンドにより意味が違ってきますが、一般的にランダム・アクセス・デバイスなどではエラー発生論理ブロック・アドレスを、シーケンシャル・アクセス・デバイスなどではコマンドで要求されたデータ量と、実際に実行できたデータ量の差 (コマンドによりブロック数であったりバイト数であったりする) を示します。すなわち、読み残しや書き残し量です。

〈追加センス長：Additional Sense Length〉

以降、何バイトのデータ (追加センス・バイト：Additional Sense バイト) があるかを示します。

〈コマンド固有情報：Command-Specific Information〉

コマンドにより定義されるフィールドですが、Copy, Compare, Copy and Verify, Search Data, Reassign Blocks をサポートしているターゲットは必ずこのフィールドをサポートする必要があります。

たとえばコピー・コマンドでは実際に動作中の装置 (コマンドを受け実行しているターゲットではない) でエラーが発生した場合、そのセンス・データの一部をこのフィールドに入れ、コピー・コマンドを受けたターゲットがコピー・コマンド発行元のイニシエータに報告します。

図15 Inquiry コマンドの CDB

ビット バイト	7	6	5	4	3	2	1	0
0	OP コード (12h)							
1	LUN			Reserved				EVPD
2	ページ・コード							
3	Reserved							
4	アロケーション長							
5	コントロール・バイト							

〈FRU コード：Field Replaceable Unit Code〉

ベンダ定義の値で、エラーの原因がどの部品か、ターゲットまたはロジカル・ユニットが自分で判断/特定できた場合、その部品コードを報告します。部品コードがどの部品に対応するかは Inquiry コマンドに対する VPD で知ることができます。

〈SKSV：Sense Key Specific Valid〉

ここが “1” になっていれば、この後に続く 〈Sense Key Specific フィールドが SCSI-2 の規約にしたがっていることを意味します。

〈センス・キーによる定義：Sense Key Specific〉

このフィールドはセンス・キーの内容によって意味が異なってきます。センス・キーが Illegal Request (不当な要求)、つまりイニシエータからのコマンドの中、またはデータ (パラメータ) の中に不適当な数値があったような場合は、このフィールドでどこが悪かったかを報告します。センス・キーが Medium Error, Recovered Error, Hardware Error の場合は、ここでリトライを何回行った結果かを報告します。また、フォーマットを実行中の場合、その最中アクセスしようとするセンス・キーは Not Ready となり、そのときこのフィールドにはフォーマットの進みぐあいを示す数値が入るので、イニシエータは、あとどのくらいかかるかを予測することもできます。

● Inquiry (12h)

このコマンドは、イニシエータがターゲットやロジカル・ユニットのデバイス・タイプや、サポートしている機能の機能レベルなど、さまざまな素姓を知るためのものです。Inquiry コマンドの CDB を図 15 に示します。この中の EVPD ビットはイネーブル・バイタル・プロダクト・データ (イネーブル VPD) の意味で、ここを “1” とした場合、その後に続くページ・フィールドでどのページの要求かを指定します。VPD はオプションですが、装置のより細かい情報を得ることが可能となっていて、ページの指定により以下の内容になっています。

ページ	VPD
00h	ページ・コード・リスト：サポートされているページ情報
01～7Fh	FRU ASCII 情報：FRU (フィールド・リプレイサブル・ユニット：Request Sense 参照) が ASCII で報告される。
80h	シリアル番号：ASCII で報告される。
81h	動作モード定義：機能レベルのコード

82h ASCII 動作モード：機能レベルが ASCII で説明される。

EVPD が “0” の場合はスタンダード・インクワイアリ・データが返されます。VPD はオプションですが、これは必須なので、図16 にそのフォーマットを示します。各フィールドの意味はつぎのとおりです。

〈クオリファイア：Peripheral Qualifier〉

この 3 ビットで、つぎに続くフィールドのデバイス・タイプのロジカル・ユニットの接続状態を示します。

〈デバイス・タイプ・コード：Peripheral Device Type〉

そのロジカル・ユニットのデバイス・タイプ・コード(1章の p.5)が示されます。

〈RMB；Removable Medium〉

“1” の場合、媒体が取りはずし可能であることを示します。

〈ISO/ECMA/ANSI バージョン〉

この装置がどの SCSI 規格のどのバージョンに適合しているかを示します。ここで SCSI-1 か SCSI-2 なのかわかります。

〈AENC〉

このビットが “1” の装置は AEN の受信機能をもったプロセッサ・デバイスであることを示します。

〈TrmIOP〉

“1” は、Terminate I/O Process をサポートしていることを示します。

〈レスポンス・データ形式；Response Data Format〉

このフィールドの値で、この Inquiry データのフォーマットがわかります。“0” は SCSI-1 形式、“1” は SCSI-2 以前のスタンダード(すなわち CCS)形式、“2” は SCSI-2 形式であることを示します。

以下の七つのビットは、それぞれの機能をサポートしているかどうかを示しています(“1”はサポートしている)。

〈RelAdr〉 リラティブ(相対)アドレス

〈WBus32〉 32 ビット・ワイド・バス

〈WBus16〉 16 ビット・ワイド・バス

〈Sync〉 同期転送

〈Linked〉 コマンド・リンク

〈CmdQue〉 タグ付きコマンド・キューイング

〈SftRe〉 ソフト・リセット

図16 スタンダード Inquiry データ

ビット バイト	7	6	5	4	3	2	1	0
0	クオリファイア			デバイス・タイプ・コード				
1	RMB	Reserved						
2	ISO バージョン		ECMA バージョン			ANSI バージョン		
3	AENC	TrmIOP	Reserved		レスポンス・データ形式			
4	追加データ長							
5	Reserved							
6	Reserved							
7	RelAdr	WBus32	WBus16	Sync	Linked	Rsvd	CmdQue	SftRe
8 15	(MSB)ベンダ ID (ASCII)(LSB)							
16 31	(MSB)プロダクト ID (ASCII)(LSB)							
32 35	(MSB)プロダクト版数 (ASCII)(LSB)							
36 55	ベンダ固有							
56 95	Reserved							
96 }	ベンダ固有							
n								

〈ベンダ ID；Vendor Identification〉 装置のメーカー名が ASCII で入ります。

〈プロダクト ID；Product Identification〉

製品名、ブランド名、モデル名などが ASCII で入ります。

〈プロダクト版数；Product Revision Level〉

製品の版数、ファームウェアのレビジョンなどが ASCII で入ります。

〈ベンダ固有；Vendor Specific〉

ベンダ固有の情報が入ることがあります。

● Mode Select/Mode Sense(15h/1Ah)

Mode Select はイニシエータが種々のモード・パラ

メータを設定するコマンド、Mode Sense はモード・パラメータを読み取るコマンドです。

では、モード・パラメータとは何でしょうか？あまりにも多岐にわたっているので一口では表しにくいのですが、しいていうと、「装置自体および媒体の内容、機能、性能を表し、または決定づける種々の数値」となります。具体的な説明に入る前に、一般的なことですが、イニシエータは Mode Select による設定を行う前に Mode Sense によってどんなパラメータになっていて、どこが変更/設定できるかを把握することが推奨されています。

さて、モード・パラメータには①カレント値、②セーブ値および③デフォルト値という三つの属性があります。カレント値は実際の動作に使われているパラメータで、Mode Select コマンドによっても変更できます。セーブ値は装置(ターゲット)に保存している値で、パワー・オン時や、リセット後の値です。この値も Mode Select コマンドの CDB 内の SP(Save Pages) ビットを“1”とすることにより、イニシエータが変更/セーブすることができます。デフォルト値は装置の工場出荷時の値で、Mode Select, SP=1 によるセーブ値がない場合、パワー・オン・リセット後に使われます。それぞれの値がどうなっているかは、Mode Sense コマンドの PC(Page Control)を指定することにより、別々に読み取り可能です。

また、ターゲットはこのモード・パラメータをすべてのロジカル・ユニットに共通にもったり、個別にもったりできますし、さらにそれらをイニシエータごとにもつこともできます(ターゲットの能力による)。イニシエータごとにパラメータをもっていない場合で、あるイニシエータに Mode Select コマンドによりパラメータを変更された場合、他のイニシエータに対してはユニット・アテンション状態を発生し、変更された旨を警告しなければなりません。

● モード・パラメータのページ

モード・パラメータは非常に多岐にわたるため、ページにより大まかに分類されます。

ページ・コード	モード・パラメータ
00h	ベンダ固有〈自由フォーマット〉
01h	デバイス・タイプ固有
02h	* ディスコネクト/リコネクト・パラメータ
03~08h	デバイス・タイプ固有
09h	* ペリフェラル・デバイス・パラメータ

0Ah	* コントロール・モード・パラメータ
0Bh~1Fh	デバイス・タイプ固有
20h~3Eh	ベンダ固有(ページ・フォーマット)
3Fh	Mode Sense コマンドで全パラメータの要求コード

*の付いているパラメータ・ページは全デバイス・タイプに共通です。ディスコネクト/リコネクト・パラメータは、ディスコネクト、リコネクトを行うタイミングを、バッファ内のデータ量/空領域や時間により管理するもので、これをうまく使うことにより、バスの使用効率を最適化することができます。

ペリフェラル・デバイス・パラメータは、ターゲットの下にどんなタイプのインターフェースを介して周辺機器(ロジカル・ユニット)が接続されているかをイニシエータが知るためのものです。インターフェースの種類は SCSI, SMD, IPI など、これはコードで示されます。

コントロール・モード・パラメータは、数々の SCSI-2 の機能を制御するもので、たとえばコマンド・キューイング機能のオン/オフや、やり方、ECA 機能のオン/オフ、AEN 機能の制御などが可能です。

● Copy(18h)

このコマンドは異なるロジカル・ユニット間、または同一ロジカル・ユニット上でデータのコピーを行います。コマンドを受けたターゲットは「コピー・マネージャ」となり、コピー元やコピー先は他のターゲットにあってもかまいません。この場合、コピー・マネージャはイニシエータとして動作し、他のターゲットに対するリードやライトを行います。

● Send Diagnostic/Receive Diagnostic Results(1Dh/1Ch)

Send Diagnostic コマンドは、ターゲットに“診断”を行わせるコマンドで、基本的にはターゲットの自己診断機能を使うものです。Receive Diagnostic Results は、診断結果を報告させるコマンドです。

● Compare(39h)

データの比較を行うコマンドで、比較するデータは同一ロジカル・ユニット上でも異なったロジカル・ユニット上でもかまいません。不一致があった場合 Check Condition で終了します。

● Copy and Verify (3Ah)

Copy コマンド同様にコピー行い、その後、コピーされたデータのベリファイ(確認)を行うコマンドです。

● Write Buffer/Read Buffer(3Bh/3Ch)

この二つはターゲットのバッファにデータを書いたり、バッファから読み出すコマンドで、ペアになっていますが、実際の使われ方はまったく違うようです。Write Buffer は、ターゲットまたはロジカル・ユニットのマイクロ・コード(ファームウェア)を更新する場合のダウン・ロードに使われることが多く、Read Buffer はエラーの発生や EOM の発生で、イニシエータから送ったデータが媒体上に書き込まれず、バッファ内に残った場合のデータのリカバリに使われることが多いようです。

● Change Definition(40h)

ターゲットやロジカル・ユニットの機能レベルを SCSI-1, SCSI-2, CCS またはベンダ固有の特殊仕様などに変更するコマンドです。

● Log Select/Log Sense(4Ch/4Dh)

ターゲットのもつ“ログ”は、バッファのオーバーラン/アンダラン発生回数や、媒体エラー、その他のエラー発生状況などが記録されている“日誌”のようなもので、これらの内容は“統計情報”といわれています。Log Select コマンドは初期値やしきい値の設定、累積の開始/停止を指示したりします。Log Sense は、統計情報の読み出し用です。また、エラー回数が設定されたしきい値を超えたとき、ユニット・アテンション状態を発生し、イニシエータに知らせることでできます。装置の予防保守や、SCSI バス使用効率のチェック/メンテナンスなどに有効なコマンドです。

■ その他のコマンドにも共通性がある

共通コマンド以外でも非常に共通性の高いコマンドがたくさんあります。この共通性をマスタすると知識に広がりが出てきます。

コマンドの OP コードに注目すると、その共通性が

よくわかります。基本的に SCSI は記憶装置に端を発していますので、コード 08h/28h の Read コマンドや 0Ah/2Ah の Write コマンドは、ほとんどのデバイスで共通になっています。

媒体に対するリード/ライトを行わないデバイスでは名称が異なるものの、たとえばプロセッサ・デバイスでは Receive/Send、通信デバイスでは Get Message/Send Message という似たような意味のコマンドになります。また、円盤系装置(HD, MO, CD-ROM など)では、コマンド体系はほとんど同じです。これらの円盤ものおよび通信デバイスは、広い意味のランダム・アクセス装置といえるでしょう。ただし、CD-ROM には、音楽専用コマンドがあり、通信デバイスには機械的動作に関するコマンド(シークやリゼロなど)がありません。

ちょっと乱暴ですが、テープ装置(長もの)、プリンタ、スキャナは広い意味でのシーケンシャル・アクセス装置にまとめることができます。ただし、プリンタはライト・オンリ、スキャナはリード・オンリです。

OP コード 01h はメカ的な“リセット”を行わせるコマンドになっているようです。円盤ものでは Rezero Unit、長ものでは Rewind 命令です。OP コード 1Bh も円盤ものでは Start/Stop Unit、長ものでは Load/Unload となっており、共通のフィーリングが得られます。

では円盤ものの 10 バイトの Seek コマンドは長ものの何になるのでしょうか? Locate という、やはりメディア上のあるアドレスに位置づけるコマンドとなります。

このように、ランダム(円盤)↔シーケンシャル(長もの)、リード/ライト用↔リードまたはライト専用、機械的動作あり↔機械的動作なし、と色々な角度からデバイス・タイプを見ると、コマンドの共通性も見えてきます。

こういう見方をしてみると、この SCSI を作った人々の目に見えない配慮がうかがえます。

おおしま・ひろたか 前出

主なSCSI装置の動作とコマンド

ハード・ディスク/CD-ROM/スキャナ/QICテープ装置を詳細に調査する

二上貴夫/大島啓孝

SCSIバスからのコマンドをよく理解し、装置を上手に活用するためには、各装置の動作やデータの記録/再生のメカニズムについて知っておく必要がある。この章では、「装置動作の理解」に焦点を置き、デバイスごとのコマンドの特徴的な使い方について説明していく。まず、SCSIデバイスの基本ともいうべきハード・ディスクを詳しく調べ、ついで、最近ニーズの高まっているCD-ROM、スキャナ、テープ装置について解説する。

(編集部)

■ SCSIデバイスの定義方法と命令構造

SCSI規格書(筆者が参照しているのはX3T9.2/375R10k)の9章から18章までは、1章ずつが異なったデバイスに関する解説、および命令・パラメータについての記述になっています。各章では、デバイスに固有の機能を生かすための命令体系が定義されています。SCSI命令8ビットのうちで、命令コードは5ビットしかないので、10種類のデバイスに対して全部独立の命令を与えることはできません。ですから、命令コードはデバイス間で同じコードを再利用しています。

たとえば命令コード1Bhは、ダイレクト・アクセス・デバイスではStart Stop Unit命令ですが、プリンタ・デバイスにとってはStop命令と解釈されます。さらにスキャナにとってはScan命令となります。

また、共通命令とメッセージ・システムはすべてのデバイスで等しく利用できます。Test Unit Readyのような一意的な命令はこの典型です。しかし、Mode Select命令などは、デバイスに固有のモードをセットできなければならないので、モード・ページに記されたパラメータ値(命令の直後にデータ・アウト・フェーズでデータとしてターゲットに送出される)によって各デバイスに固有の機能を指定します。

1デバイスのMode Selectに対してモード・ページは複数あります。そして、モード・ページを指定するモード・ページ・コードはSCSI命令コードと同じくデバイス間で再利用されています。たとえばダイレクト・アクセス・デバイスのモード・ページ・コード03hは、フォーマット・デバイス・ページですが、スキャナでは03hは測定単位ページを示します。

このへんは初心者にとって、すこし複雑かもしれませんが、しかし、命令やパラメータ・リストの構造は論理的に統一されていますし、第2章の最後でもふれていましたが、異種デバイスであっても似た機能には同じ命令やページ・コードを振るような努力がはらわれています。したがって、一つのデバイスで規格の読み方をマスタしてしまうと、あとはそれからの類推がきくので、理解がスピーディになります。

■ 命令に付随するパラメータ・リストの構造

SCSI命令の構造は比較的簡単です。すなわち6, 10, 12バイトの直列データですから、表を手に入れるか自分で作成すれば一目瞭然です。しかし、命令に付随するパラメータは、少し構造が込み入っているので、命令の詳細を説明するときに、おのおのについて具体的に示していきます。

命令に付くパラメータとは、命令バイトでは表現できない可変長の長いデータ列や、一つの命令についてくるタイプの異なるパラメータ群のことを意味します。SCSIではこの群のことをページと呼びます。

3.1

ダイレクト・アクセス・デバイス

二上 貴夫

ダイレクト・アクセス・デバイスは SCSI の起源ともいえるモデルですから、規格書ではこれに 72 ページと最多のページが割かれています。元来はハード・ディスクがダイレクト・アクセス・デバイスであったわけですが、一部のテープやフロッピー、大容量メモリ・デバイスなどもこのカテゴリに入られています。

■ ブロック・デバイス

これらすべてに共通するデバイスの性質は、一定の長さのデータをまとめて記憶することです。このまとまりをブロックと呼びます。ブロックの記憶(メディアへの書き込み)は、記憶させたいデータに加えて特定の場所(ブロック・アドレス)をホストが指定することで実現されます。ホストはこのブロック・アドレスを忘れないかぎり、いつでも以前書き込んだデータを読み出すことができるわけです。

この機能がなければ、今日の汎用コンピュータは成立しなかったといってもよいくらいの基幹技術ですから、いまさらと思われるかもしれませんが、SCSI のダイレクト・アクセスの定義を明確にするためには重要です。いいかえるとダイレクト・アクセス・デバイスは、任意のアドレスに定型長のデータをリード/ライトできることが生命線なのです。

■ メディア交換型か非交換型か

つぎに忘れてならないのは、メディアを交換できるデバイスも同類として扱う場合があることです。このようなデバイスではホストが先ほどのブロック・アドレスを記憶していたとしても、誰かが勝手にメディアを入れ替えてしまったならば、同じブロック・アドレスに全然違うデータが入っているのですから、困ったことになります。たとえば、カセット型のハード・ディスクや大容量フロッピー、あるいは MO などはこの部類です。こうした場合はいくつかの補足手段をこうじて使用することになります。

本稿ではダイレクト・アクセス・デバイスという名前はあまりに長すぎるので、以降、ディスクと省略することにします。物理形態はどうであれ、論理的な使用法と特徴がディスクと同じものならばすべてディス

クと呼んでしまおうというわけです。

1

ディスク用 SCSI 命令と三つの立場

リード/ライトさえできればよいのがディスクであるとはいっても、現実にはいろいろなしぐらがあり、それだけではすみません。ディスク用 SCSI 命令が 27 もあるのはこのいろいろな問題を解決するためでもあるのです。ここでは、それらをしぐら別に整理して、こんなときに使うのだというような解説を行います。もちろん、ここで説明する使い方がすべてというわけではありませんが、SCSI 命令を理解するには効果的な方法だと思います。

まず、ディスクにかかわる人の立場からながめて三つの切り口を考えます。

- (1) ディスクを開発製造する立場(メーカー)
- (2) ディスクを記憶装置として利用する立場(ユーザ)
- (3) ユーザがディスクを利用できるように支援する立場(サポータ)

メーカーとユーザは一般には境界が明確であると考えられます。一方、ユーザとサポータは、小規模なシステムでは一人二役の場合も多いでしょう。それでも概念としては両者の区別は明らかなです。ここでは、システムの立ち上げすら知らない、純粋にパッケージ・ソフトを使うだけの人をユーザと呼びましょう。そしてサポータはユーザの见えないところでシステムを維持する人としします。こうしておいて誰がどのような SCSI 命令を使うのかを考えます(表 1)。

■ メーカーの立場

ディスクを作った本人ですから当然すべての命令を実行します。ただし、その多くはディスクの性能を試すための実行であって、ディスクの利用目的そのものの使い方ではありません。メーカーが自身のために使う SCSI 命令は、表 1 に内部デバッグ系として切り出した Read Long と Write Long 命令です。これらは、記憶したいデータをリード/ライトすると同時に、その記

表1 ハード・ディスク用カテゴリ別 SCSI 命令一覧

カテゴリ・マーク			機能系列	該当する SCSI 命令
M	U	S		
*			内部デバッグ	Read Long, Write Long
T		*	初期設定	[Mode Select], Format Unit, Write Same
T		*	使用準備	[Test Unit, Inquiry, Mode Sense], Read Capacity, Start Stop Unit
T		*	メディア管理	Rezero Unit, Set Limits
T		*	欠陥処理	Read Defect Data, Reassign Blocks, Verify
T	*		リード/ライト	Read 6/10, Write 6/10, Write and Verify
T	*		マルチユーザ	Privent Allow Medium Removal, Release, Reserve
T	*		スループット	Pre-Fetch, Lock-Unlock Cache, Synchronize Cache, Seek
T	*		データ検索	Search Data Equal, Search Data High, Search Data Low

M:メーカー U:ユーザ S:サポータ *:おもに使用する立場 T:テスト的に使う立場

SCSI のディスク(ダイレクト・アクセス)命令は上記のように使用目的に対応して類型化するとわかりやすい。[] 内は共通命令だが、重要度が高いためにのせた

録誤りを補正するための ECC コードも、ホストへ転送/ホストから転送、する命令です。いわばメディアから SCSI のポートまでの内部テストとデバッグに用いられる命令です。

ディスクの購買側がその性能評価のためにこれらの命令を使用することもあります。このような場合は、購買側もメーカーの延長であって、メディアとしてディスクを利用しているわけではありません。データ・キャッシュに関してもその機能を使わないといった機能設定(Read 命令の FUA ビット)は、一般のリード/ライトというよりも開発やテストのための機能と考えたほうがわかりやすいでしょう。

また、ディスクにかぎりませんが、Identify メッセージの LUN TAR ビットをセットして送られる命令やデータはデバイスのターゲット・ルーチンへ送られます。これは、ディスクなどの論理装置の保守に用いられるものなのでメーカーの専用と考えます。さらに、メーカーによってはベンダ・ユニーク命令を独自に定義して保守用途に使っている場合もあります。ディスクのシリアル番号やデフォルトの値をメディアのユーザからは見えない領域に書き込む命令などがそれです。もちろんこうした機能に対しては、ハードウェアのジャンパなどで二重に保護されています。ですから、ユーザが誤ってその命令と等しい命令コードを送っても事故にはなりません。

■ ユーザの立場

パッケージ・ソフトを利用するだけのユーザは SCSI など意識しないわけですが、その裏でパッケージ・ソフト自体は OS を経由してさまざまな SCSI 命令を発行します。この場合はリード/ライト系の命令が多用されるのは目的からいって明らかです。もし、OS と I/O ドライバが高機能であって、SCSI につながる外部記憶との I/O レートをより高くする機能が実装されていれば、スループット系の命令も発行されるでしょう。さらに、ユーザの使うマシンがマルチユーザ・システムであれば、マルチユーザ系の命令を自動発行する可能性もあります。また、ディスクにかぎりませんが、コマンド・キューイングが使えるデバイスに対しては、キューイング・タグを送ってマルチユーザ環境でのスループットを向上させるという SCSI-2 のもっとも芸の込んだリード/ライトを実行することになるでしょう。残念ながらここまでのレベルを実現しているシステムの事例を筆者はまだ知りません。

もう一つ、機能から考えるとユーザのための命令ですが、使われているところを見たことがないのがデータ検索系の命令です。本来、ホストが行うべきデータ内容の検索をディスクのインテリジェンスを利用して行わせるというものです。どういう経緯でこの系列の命令が規格になったのかは知りませんが、この命令系が多用されるのはしばらく先になるでしょう。

表2 Mode Selectのディスク固有ページ要約

ページ	設定項目
08h	キャッシュ(プリフェッチのサイズや制御など)
05h	フレキシブル・ディスク(フロッピー・ディスクの転送レート、シリンダ数など)
03h	フォーマット・デバイス(ゾーン当たりのトラック数、代替セクタやトラックの数など)
0Bh	使用可能なメディア・タイプ(最大4種まで指定できる)
0Ch	ノッチ(シリンダ当たりのセクタ数を可変にした場合の領域)
01h	リード・ライト・エラー・リカバリ(リトライ・カウントなどエラー回復方法)
04h	リジッド・ディスク・ジオメトリ(ハード・ディスクのシリンダ数、回転数など)
07h	ベリファイ・エラー・リカバリ(ベリファイ付き命令のリトライ・カウントなど)

■ サポートの立場

サポートとしてすべての SCSI 命令を意識してユーザの支援を行っているわけではありません。しかし、サポートの業務の結果として、ディスクに対して特定のモードでアクセスすることになります。その典型は、初めてディスクをシステムへつなぎ込んだときに行う初期設定系命令(Format 命令と Mode Select 命令)でしょう。細かいパラメータの値は、システム任せになりますが、命令の実施を決めるのはサポートです。

また、メディア全体ではなく一部についてデータを初期化したい場合には、Write Same 命令が使えます。さらに、システムに電源を入れて立ち上げる際には使用準備系の命令が自動または手動で発行されます。

ディスクの宿命であるメディア欠陥の発見や処理もサポートが OS のユーティリティを通じて実施するはずで、これは、結果として欠陥処理系の SCSI 命令になってディスクへ到達します。

表2 にディスク固有のページの要約を示します。

● 使用準備

- ▷ Read Capacity：ディスクの容量(ブロック数とブロック長)を返す。
- ▷ Start Stop Unit：メディアへのアクセス権、ロード/イジェクトを制御する。

● メディア管理

- ▷ Rezero Unit：ディスクをメーカー定義の特定状態にする(ヘッドをトラック0へ移動するなど)。
- ▷ Set Limits：リード/ライトのブロック・アドレス、アクセス制限を設ける。

● 欠陥処理

- ▷ Read Defect Data：欠陥記述データをホストへ返す。
- ▷ Reassign Blocks：ホストからディスクに特定ブロックを欠陥とみなして代替ブロックを割り当てるよう指示する。
- ▷ Verify：メディアに記録されているデータをベリファイする。

● リード/ライト

- ▷ Read(6 バイト命令)：ブロック・アドレス 16 ビット以下で転送ブロック数 255 までのブロック・データを読む。
- ▷ Read(10 バイト命令)：ブロック・アドレス 32 ビット以下で転送ブロック数 65536 までのブロック・データを読む。
- ▷ Write(6 バイト命令)：ブロック・アドレス 16 ビット以下で転送ブロック数 255 までのブロック・データ

2

機能別ディスク命令の解説

つづいて各系列での命令を概説します。

● 初期設定

- ▷ Format Unit：ディスクのフォーマットを実行する。フォーマットには同時にインタリーブなど基本構造の指定と欠陥補償にかかわるデータを与える。
- ▷ Write Same：同一データを複数のブロックへくり返し書き込む。
- ▷ Mode Select：動作モードを指定するパラメータを設定する。設定は、ページ単位に行う。

を書く。

▷ Read(10 バイト命令)：ブロック・アドレス 32 ビット以下で転送ブロック数 65536 までのブロック・データを書く。

▷ Write and Verify：10 バイト Write 命令と同じ命令で同時にベリファイを行う。

● マルチユーザ

▷ Privent Allow Medium Removal：メディアの取り外しを許可/禁止する。

▷ Release, Reserve：「ディスクの専有/専有の解除」を行う。

● スループット

▷ Pre-Fetch：キャッシュへのプリフェッチを行う。

▷ Lock-Unlock Cache：キャッシュのロック/解除を

行う。

▷ Synchronize Cache：キャッシュとメディアのデータを一致させる。

▷ Seek(6 バイト命令)：ブロック・アドレス 16 ビット以下の領域でシークを行う。

▷ Seek(10 バイト命令)：ブロック・アドレス 32 ビット以下の領域でシークを行う。

● データ検索

▷ Search Data：ブロック内をさらにレコードの集合として同値検索。

▷ Search Data High：ブロック内をさらにレコードの集合として非同値検索。

▷ Search Data Low：ブロック内をさらにレコードの集合として非同値検索。

ふたがみ・たかお (株)東陽テクニカ エレクトロニクス事業部 開発部

3.2

CD-ROM デバイス

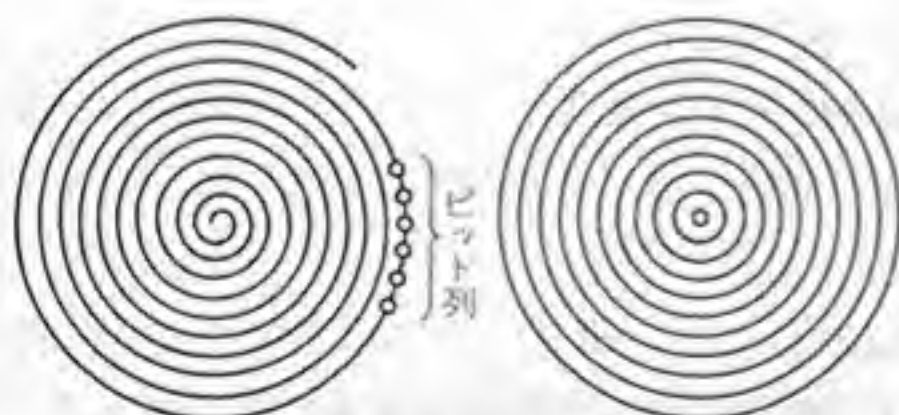
二上 貴夫

1

CD の分類

一口に CD とはいうものの、最近は多種多様な名前があって分類するのに苦労します。CD という名前にくくれる同族に新種が続きつぎと登場しているからです。文献によって名前と実体が異なるので、以下はそれらの最大公約数的な分類学と思ってください。

図1 CD と HD の記録構造の違い



(a) CD-ROM 渦巻の総延長は 12cm 盤で 約 5.3km、容量は 750M バイト
(b) FD, HD, MO などは 同心円

■ 書き込み能力による分類

もともと CD は音楽用レコードのデジタル版だったわけですが、最近は書き込みが 1 回だけ可能なメディアとドライブが利用できます。よって CD は、書き込みができるか否かで 2 種類に分けられます。

(1) 読み出し専用 CD

たんに CD もしくは CD-ROM と呼ばれています。いわゆる市販の音楽用 CD やパッケージ・ソフトの頒布用として利用されているものです。

LSI の製造プロセスと同じ要領でフォト・エッチングにより原盤を作成します。これから金属の母型を起こします。母型はプラスチック製品の射出成形に使う金型と似た役目をはたします。つまり、母型にポリカーボネート基材を注型し、透明円盤を作るのです。つぎに、この円盤に反射用アルミ層を蒸着してからアルミを保護する表面処理をしてできあがりです。データは、図 1 に示すように渦巻状になったデータ・トラックにそって、ビットと呼ばれる深さ 120 nm の穴とランドとのレーザ光反射率の差でビットの “1”/ “0” を記録しています。

データの読み出しと音楽再生は、螺旋の内側から外へ向けてこのビット列を線速度一定(1.2 から 1.4 m/

秒)で追いかけることで実現されます。

(2) 書き込み可能 CD

CD-R (CD-Recordable) は、1 回だけ書き込める CD です。ただし、消去はできません。メディアの形状は CD-ROM と同じですし、読み出し特性も同じです。ポリカーボネート基材に有機色素をコーティングした上に金の反射膜を蒸着しています。未使用の CD-R に書き込みを行うときには読み出しに比べて高い出力のレーザーを照射します。すると色素層が変成して反射率が変わります。この反射率の変化の「あり/なし」が CD-ROM と同じ結果をうむので、読み出しは CD-ROM とまったく同じになります。

SCSI では CD-ROM を定義しているのですが、以上のようなわけで CD-ROM と CD-R を読むことは CD ドライブのユーザとしては、差別する必要がありません。CD-R の書き込み装置に関しては SCSI での定義はありませんが、市販されている書き込み装置は SCSI を使っているものが多いようです。また、CD-R では、複数回に分けた書き込みを許すためにマルチセッションというデータ管理方法を定義しています。これは、今のところ SCSI には入っていません。

■ 使用用途による分類

つぎに、音楽用 CD とコンピュータ用 CD の違いを考えましょう。規格書では、データ用 CD を CD-ROM、音楽用を CD-DA (Digital Audio) と呼んでいます。ここで音楽用というのは、本来のハイファイ再生を行うためのものを意味しており、マルチメディアのアプリケーションは除外されます(マルチメディアの場合は、データ領域にコード圧縮されたうえで記録された音声データを伸長、再生している)。もう少し正確にいうと CD-ROM は CD-DA の規格にのった規格です。ですから、両者が混在しているメディアも CD-ROM と呼びます。

さらにコンピュータ用として CD-ROM のデータ構造をベースとして、その上位構造を定義したものに CD-I、CD-XA などがあります。SCSI 規格書 (Rev.10k) ではこれらについて言及していないので、本稿ではそのフォーマットについては説明しません。

■ サイズによる分類

従来の直径 12 cm のものと 8 cm のミニディスクと呼ばれる小型の盤の 2 種類が市販されています。8 cm

版は、12 cm と物理的な構造はまったく同じで、螺旋の外側がないだけの小容量版と考えられます。

2

CD のデータ構造

■ 論理的なデータ構造

CD は、リアルタイムの音楽再生とデータの再生の両方を可能にしているところから、従来のハード・ディスクやフロッピー・ディスクとは著しく異なる部分があります。このデータ構造が CD-ROM を理解するにはもっとも重要です。もう一度、図 1 を見てください。CD-ROM に記録されているデータは、円柱に縄を内側から外へ向けてぐるぐる巻き付けた円盤のようになっています。ですから、ディスクのようにシリンダが同心円状に取り巻いている構造とは異なります。

この縄状の 1 次元データ列はいくつかのやり方で構造化されています。そのもっとも基本的な構造がトラック/インデックス/セクタ構造です(図 2 の ERD を参照)。

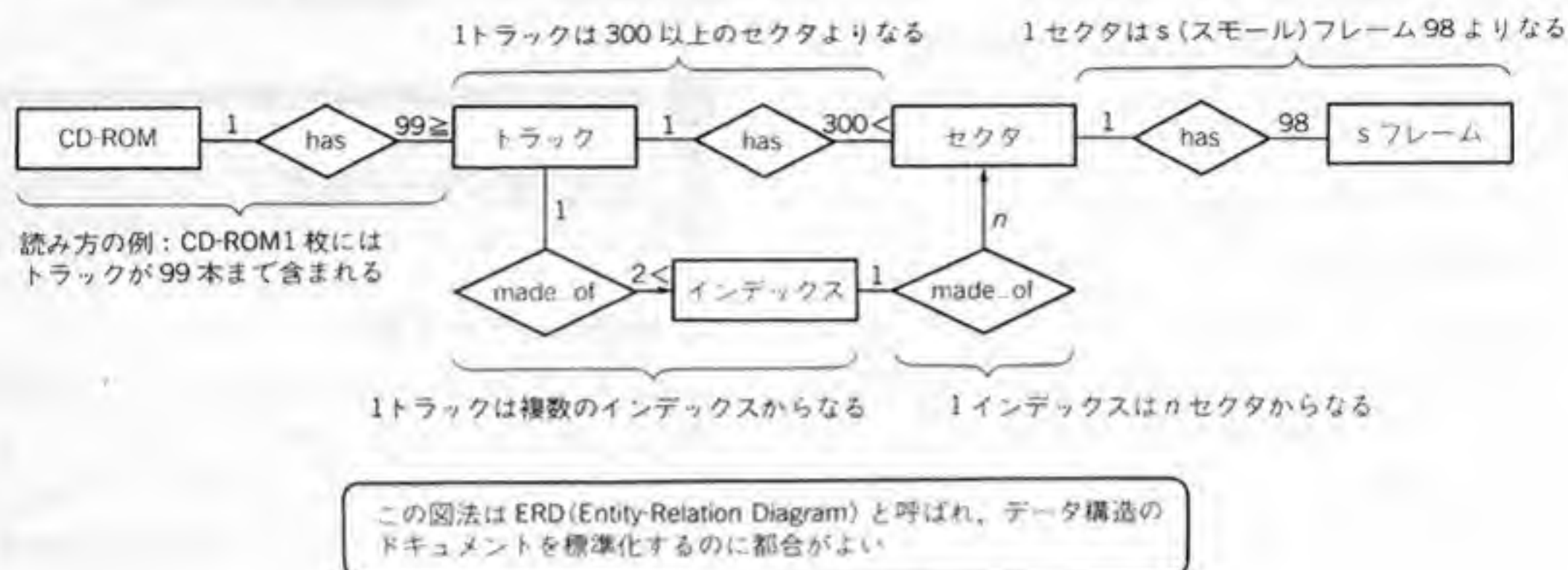
● トラック

CD-ROM の渦巻状になっている縄の内側をつまんで左から右へ開いたのが図 3 です。縄、すなわち CD-ROM は、いくつかのトラックの集合として定義されます。ここでいうトラックは、磁気ディスクのトラックとは定義がまったく異なります。磁気ディスクのトラックは物理的な円周に対応しています。そして、トラックの記憶量はメディアによって一意に決まってしまうのに対し、CD-ROM のトラックは、アプリケーションの区切りを意味します。

たとえば、音楽ならば頭出しの空白に加えて歌 1 曲が 1 トラックです。データでは、磁気ディスクのボリューム 1 本に相当します。CD-ROM の先頭と末尾のトラックは、それぞれリード・イン・エリア、リード・アウト・エリアと呼ばれています。

リード・イン・エリアは、トラック番号が 00h (00h BCD) です。この領域にはユーザは直接アクセスできません。ここには、TOC (後述) が記録されています。リード・アウト・エリアは、トラック番号 AAh でやはりユーザはアクセスできません。この 00h と AAh の間がユーザの使用できるトラックです。ユーザのト

図2 ERD 記法による CD のトラック/インデックス/セクタ構造



トラックは、1個から最大99個まで定義されています。トラック番号は1以上99以下の任意の数字から始まり、1ずつ増加します。トラックの最低長は、後述のセクタ数で最低300セクタ以上あることと規定されています。

● インデックス

一つのトラックは複数のインデックスからなっています(図4)。トラックの先頭は無音領域、データならば空白となっているトランジション・エリアです。ここには、インデックス番号として0が振られます。これに続いて実際の曲やユーザのデータが記録されているインフォメーション・エリアが続きます。ここには、インデックス番号1から最大99までのインデックスが振られます。

以上でわかるとおり、トラックとインデックスをどのように振るかはアプリケーションに依存します。

● セクタ

インデックスの内部は、セクタで構成されています(図5)。セクタはSCSIからアクセスできる最低のユーザ・データ単位です。セクタには、ユーザのデータが記録されているメイン・チャンネル(全部で2352バイト)と補助データの入っているサブチャンネル(98ビット×8)からなっています。規格書ではこのへんのデータ構造の用語が構造化されて解説されていません。ここでは、できるだけ整理した形の用語を用いることに

図3 CD-ROMのトラック



します。

● セクタ・メイン・チャンネル(ブロック・データ)

CD-ROMとしてのユーザ・データの記録方法にはモード1とモード2の2種類があって、1セクタの利用法が異なります。

モード1では、ユーザ・データは2048バイトです。モード2では、モード1の2048バイトに加えて図5のレイヤード ECC(データの誤り補正を行うためのコード)と記された領域もユーザ・データ域として利用するので、合計2336バイトが1セクタ当たりのデータになります。

一方、CD-DAとしてのセクタの記録は、2336バイトにステレオ左右の音データがおのおの16ビット長で記録されます。すなわち、1セクタ当たり588の音声標

図4 トラックとインデックスの構造

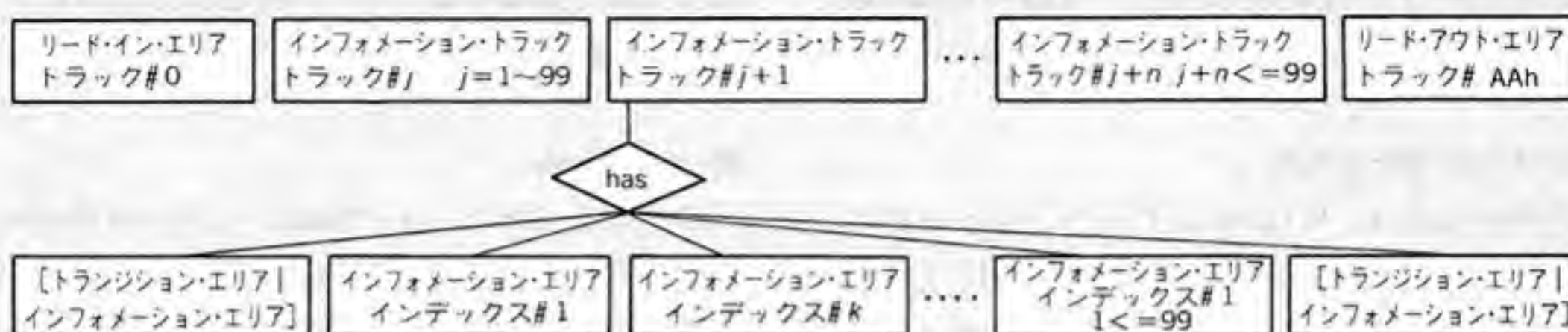


図5 セクタの論理構造(サブチャンネルはQ以外は省略した)

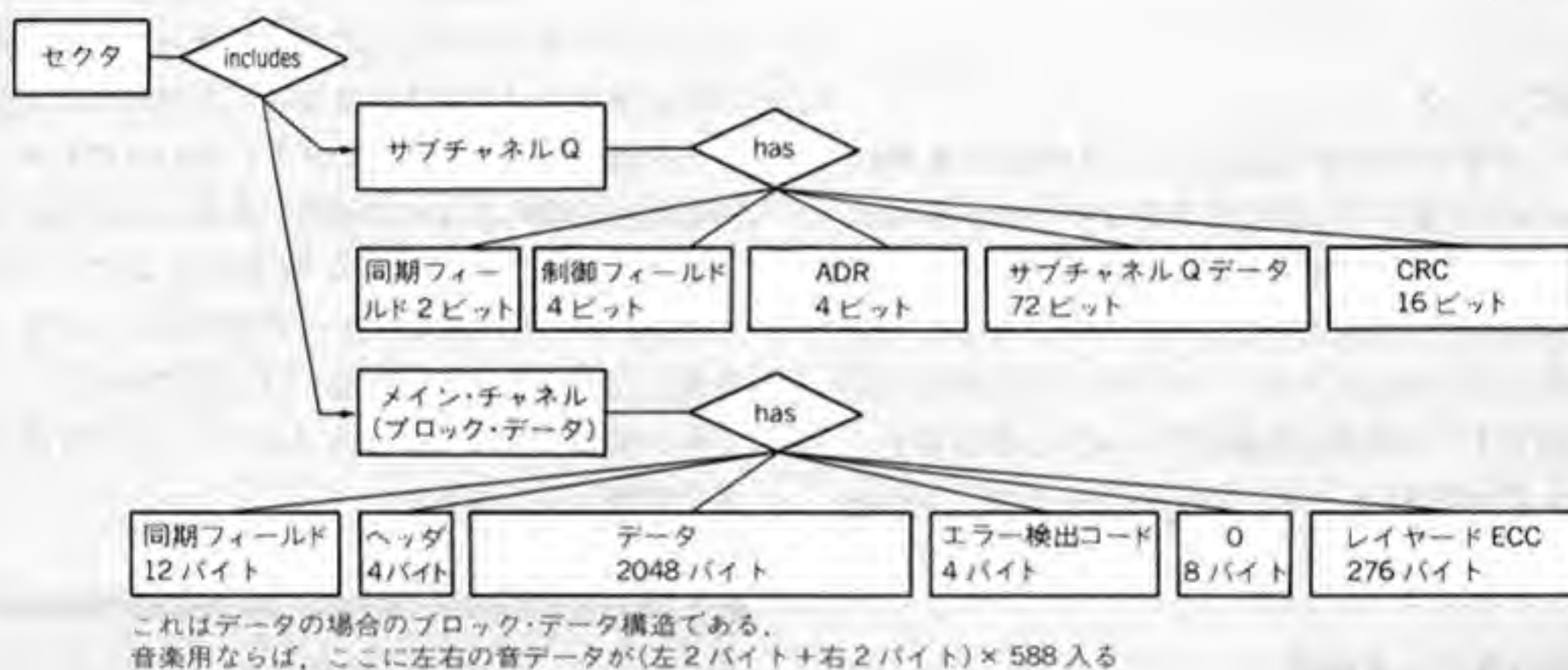
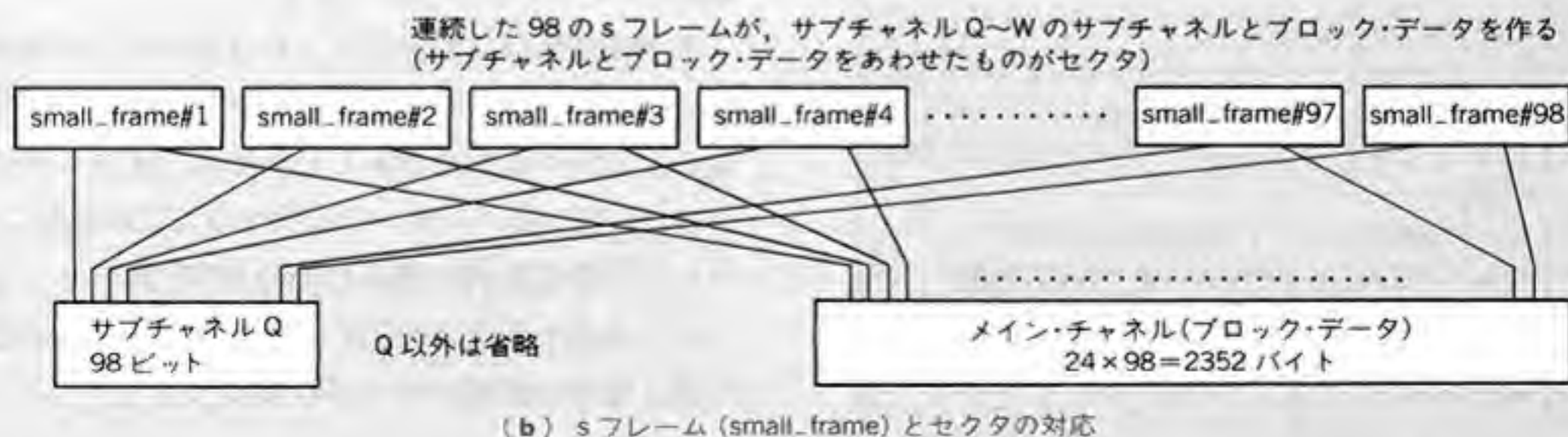
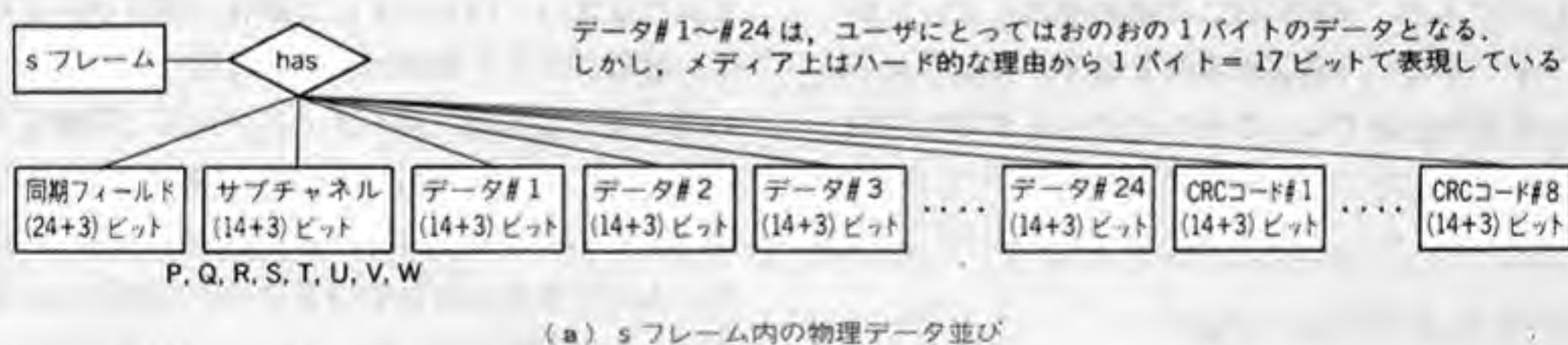


図6 Qチャンネルの構造



本点があります。CD-DA の標本化周波数は 44.1 kHz なので、1 セクタには $588/44100=1/75$ 秒の音楽が録音されています。

● セクタ・サブチャネル

サブチャネルは、P, Q, R, S, T, U, V, W と命名され、おのおのセクタ当たり 98 ビットあります。CD で現在使用されているのは P と Q のみです。P チャネルはミューティング制御とトラック区切りを示します。Q チャネルには、各種の制御データが入っています(図 6)。Q チャネルの 98 ビットの内訳は表 3 のようになっています。

● 論理ブロック

トラックからセクタまでのデータ構造と並列に SCSI からアクセスする側の都合がよいデータ単位として論理ブロックが定義されています。ハード・ディスクと同様に一つのセクタを二つのブロックとして分割したり、二つのセクタを一つのブロックとみなすことが可能です。この単位を論理ブロックと呼びます。1 ブロックの長さ(ユーザ・データの長さ)は、Mode Select 命令で与えられます。

■ 物理的なデータ構造

トラック、インデックス、セクタは、SCSI を通して外から見える論理的な CD データ構造でした。この構造を構成しているのが物理的な穴の列すなわちビットだったわけですが、実際にはこの論理構造とビット列の間にフレームという構造が存在します。このフレームという言葉は CD では、さきほどのセクタの代名詞

表 3 Q チャネルの 98 ビットの内訳

ビット	内 容
2	サブチャネル同期フィールド
4	コントロール・フィールド(セクタにあるデータの型を示す。音声、データの識別など)
4	ADR フィールド(この Q チャネルにあるデータの型を示す)
8	セクタの属するトラック番号(BCD 表現)
8	セクタの属するインデックス番号(BCD 表現)
24	トラック相対 MSF アドレス
8	予約
24	絶対 MSF アドレス
16	CRC エラー検出コード

としても使われているので紛らわしくなっています。ここでは、物理的なビットの集合のことを s フレーム (Small frame の短縮形) と統一して呼ぶことにします。

● s フレーム

セクタは、588 ビットの物理データを 98 個接続したうえでデータ変換を施した結果として得られます。s フレームは、図 6 のように 24 バイトのメイン・チャネルのデータとサブチャネル・データに同期ビット、エラー補正コードを付加したまとまりです。これは、CD の最終単位で、このビットは、銀色の盤面に作られた穴の一つ一つに対応しています。幅 0.6×長さ 1~3 ミクロンほどの小さな穴の列ですから、いつも正しく読み出せるわけではありません。そのために各種のエラーを補正する技術が使われているわけですが、本稿の目的とは離れますので省略します。ここでは、588 ビットの連続した s フレーム 98 本をそこにマークされた Sync ビットで同期をとって読み出し、エラー補正を施した結果として、2352 バイトのブロック・データ 1 本と 98 ビットのサブチャネル・データが 8 本作られると理解してください。

3 アドレッシングについて

メディアの先頭トラックから 1 セクタずつ数え上げてゆけばすべてのセクタには固有の番号が与えられます。論理ブロック長がセクタ長に等しいときには、この固有番号は論理ブロック・アドレスと等価です。ここまでは、ハード・ディスクと大差ありませんが、CD-ROM では、それに加えて MSF というデータがあらこちらに書き込まれています。この MSF は、先頭セクタからの距離を分、秒、フレーム(1/75 秒)の三つの BCD コードで示したものです。

すでに読者は、CD-ROM 上の任意のセクタを指定するには場合によっては複数のアドレッシングが可能であることに気づかれたかもしれません。まさにそのとおりで、つぎのセクションで説明する SCSI 命令には、アドレスの指定形式を選ぶための MSF 指定ビットやアドレス方法の数だけ SCSI 命令があるなど多少乱雑とも思える命令体系になっています。

4

データへのアクセスと SCSI 命令概要

CD-ROM のターゲットに実装される SCSI 命令は、つぎの 3 種類からなっています。

(1) メディア全体の情報を取得したり、論理ブロックのサイズを決めるといった設定にかかわる命令

Read Sub-Channel, Read TOC

(2) CD-DA へのアクセス(オーディオ系)

Pause/Resume, Play Audio(10, 12), Play Audio MSF, Play Audio Track/Index, Play Track Relative(10, 12)

(3) CD-ROM のアクセス(データ系)

Read Header, Read CD-ROM Capacity

蛇足ですが Write 命令は存在しません。また、Read 命令の定義はディスクと同等です。

図7 Read Header 命令

ビット		7	6	5	4	3	2	1	0
バイト									
0		OP コード (44h)							
1		LUN			予約			↓	予約
2		論理ブロック・アドレス							
3									
4									
5									
6		予約							
7		転送長							
8									
9		コントロール							

MSF 指定

5

CD-ROM 専用命令の説明

▷ Read Header 命令の詳細

命令で指定された論理ブロックのヘッダ(図5参照)をホストに返します(図7)。

MSF 指定：本ビットが1ならば、ヘッダ・データ中の絶対 CD-ROM アドレスは MSF データとなります。逆に0ならば、論理ブロック・アドレスとなります。

ヘッダ・データはつぎのとおりです。

バイト	内容
0	CD-ROM データ・モード
1~3	予約
4~7	絶対 CD-ROM アドレス

<ヘッダ・データの項目説明>

「CD-ROM データ・モード」は、指定の論理ブロックの内容の組み合わせ(モード 0, 1, 2 のいずれか)を示します。

コード	ユーザ・データ領域	補助領域
00h	すべてのバイトが0である	すべてのバイトが0である
01h	ユーザ・データが記録されている	エラー補正(LECC)が記録されている
02h	ユーザ・データが記録されている	ユーザ・データが記録されている

▷ Read Sub-Channel 命令

現在位置のサブチャネルのデータをホストへ返します(図8)。

「SubQ」ビットが1ならば、データ・イン・フェーズで Q チャネルのデータがホストへ送られます。0 ならばサブチャネル・データを送りません。

「サブチャネル・データ・フォーマット」は、フォーマット・コードによって、以下の返されるデータのフォーマットを指定します。

コード	返されるデータ
00h	サブ Q チャネル・データ
01h	CD-ROM の現在位置
02h	メディア・カタログ番号
03h	トラック国際標準録音コード (ISRC)
04~EFh	予約
F0~FFh	ベンダ固有

「トラック番号」は、フォーマット・コードが 03h の場合にのみ有効で、この番号で指定されたトラックの ISRC を返します。

▷ CD-ROM 現在位置フォーマット

バイト・オフセット 4 のコードが 01h となることとデータの 16~47 がないほかはサブ Q チャネル・データ・フォーマットと同じです。その他のフォーマットは省略します。

図8 Read Sub-Channel 命令

		MSF 指定							
ビット バイト	7	6	5	4	3	2	1	0	
0	OP コード (42h)								
1	LUN			予約			↓	予約	
2	予約	SubQ	予約						
3	サブチャネル・データ・フォーマット								
4	予約								
5									
6	トラック番号								
7	転送長								
8									
9	コントロール								

図9 Read TOC 命令

ビット バイト	7	6	5	4	3	2	1	0
0	OP コード (43h)							
1	LUN		予約				予約	
2	予約							
3								
4								
5								
6	開始トラック							
7	転送長							
8								
9	コントロール							

図8-1 サブQ チャネル・データ・フォーマット

ビット バイト	7	6	5	4	3	2	1	0
0	予約							
1	オーディオ・ステータス							
2	サブチャネル・データ長							
3								
4	サブチャネル・データ・フォーマット・コード (00h)							
5	ADR				コントロール			
6	トラック番号							
7	インデックス番号							
8 } 11	絶対 CD-ROM アドレス							
12 } 15	トラック相対 CD-ROM アドレス							
16	McVal	予約						
17 } 31	メディア・カタログ番号 (UPC/バーコード)							
32	TcVal	予約						
33 } 47	国際標準録音コード (ISRC)							

ステータス・コード	説明
00h	オーディオ・ステータスに対応していないか有効でない
11h	オーディオ再生中
12h	オーディオ一時停止中
13h	オーディオ再生終了
14h	オーディオ再生がエラー終了した
15h	返すべきオーディオ・ステータスがない

ADR コード	説明
0h	モード情報は記録されていない
1h	サブチャネルQには現在の位置データがコード化されている
2h	サブチャネルQにはメディア・カタログ番号がコード化されている
3h	サブチャネルQにはISRCがコード化されている

ビット位置	"0"	"1"
0	プリエンファシス	プリエンファシス
1	デジタル・コピー禁止	デジタル・コピー可
2	オーディオ・トラック	データ・トラック
3	2チャンネル・オーディオ	4チャンネル・オーディオ

McVal ビットが1ならば、データ中のメディア・カタログ番号は有効。
TcVal ビットが1ならば、トラック ISRC は有効。

▷ Read TOC 命令の詳細

TOC (Table Of Contents: トラック情報テーブル) をホストへ返します(図9)。


図9の「開始トラック」は、TOCとしてホストへ返すトラック番号の最少の値を指定します。この値が0ならば、最初のトラックからのテーブルが返されます。

TOC データ・フォーマットはつぎのとおりです。

バイト	内容
0-1	TOC データ長
2	先頭トラック番号
3	末尾トラック番号
(0-7)	TOC トラック記述域)
0	予約
1, [7-4]*	ADR
1, [3-0]	コントロール
2	トラック番号
3	予約
4-7	絶対 CD-ROM アドレス

- ・ADR: この TOC エントリが見つかったところの ADR 値
- ・コントロール: この TOC エントリが見つかったところのコントロール値
- ・トラック番号: 本 TOC 記述が有効なトラック番号
- * 1 バイト目の 7-4 ビットを表す(以下同様)

図10 Pause/Resume 命令

ビット バイト	7	6	5	4	3	2	1	0
0	OP コード (4Bh)							
1	LUN			予約				
2	予約							
3								
4								
5								
6								
7								
8	予約							
9	コントロール							

Resume :

値	動作
0	現在のブロック再生が終わるとホールド・トラック状態になる。
1	ポーズを終了し、つぎのブロックからの再生を開始する。

▷ Pause/Resume 命令の詳細

オーディオ再生を停止/再開します。本命令は、オーディオ再生命令の Immediate ビットを 1 とセットして実行したときに組み合わせて使用します。この命令は重複して使われてもエラーとはしません(図10)。

▷ Play Audio (10, 12) 命令の詳細

指定の論理ブロックから再生を実行します。指定ブロックが音楽領域でない場合にはエラーを返します(図11)。

▷ Play Audio MSF 命令

指定された開始 MSF から終了 MSF まで再生を実行します(図12)。

▷ Play Audio Track/Index 命令の詳細

指定された開始トラックとインデックスから終了トラックとインデックスまで再生を実行します(図13)。

▷ Play Track Relative (10, 12) 命令の詳細

開始トラックで指定したトラックを基点に 2 の補数で表現されたトラック相対論理ブロック・アドレスだけオフセットした位置から再生を実行します(図14)。

▷ Read CD-ROM Capacity 命令

論理装置の容量をホストへ返します(図15)。

PMI ビットが 0 のときには「論理ブロック・アドレス」は 0 とします。

図11 Play Audio (10) 命令 (Play Audio (12) は省略)

ビット バイト	7	6	5	4	3	2	1	0	
0	OP コード (45h)								
1	予約			LUN				RelAdr 	
2	再生開始する論理ブロック・アドレス								
3									
4									
5									
6	予約								
7	連続的に再生する論理ブロックの数								
8									
9	コントロール								

図12 Play Audio MSF 命令

ビット バイト	7	6	5	4	3	2	1	0
0	OP コード (47h)							
1	LUN			予約				
2	予約							
3	開始Mフィールド							
4	開始Sフィールド							
5	開始Fフィールド							
6	終了Mフィールド							
7	終了Sフィールド							
8	終了Fフィールド							
9	コントロール							

図13 Play Audio Track/Index 命令

ビット バイト	7	6	5	4	3	2	1	0
0	OP コード (48 h)							
1	LUN			予約				
2	予約							
3								
4	開始トラック							
5	開始インデックス							
6	予約							
7	終了トラック							
8	終了インデックス							
9	コントロール							

図14 Play Track Relative (10)
(Play Track Relative (12)は省略)

ビット バイト	7	6	5	4	3	2	1	0
0	OP コード (49h)							
1	LUN			予約				
2	再生開始する論理ブロック・アドレス (トラック相対)							
3								
4								
5								
6	開始トラック							
7	連続的に再生する論理ブロックの数							
8								
9	コントロール							

図15 Read CD-ROM Capacity 命令

ビット バイト	7	6	5	4	3	2	1	0	
0	OP コード (25h)								
1	LUN			予約				RelAdr	
2	論理ブロック・アドレス								
3									
4									
5	予約								
6									
7									
8	予約						PMI		
9	コントロール								

値	動作
0	Seek 命令で探せる最大の論理ブロック・アドレスを返す。
1	命令で指定された論理ブロック・アドレス以上のアドレスにおいてデータ転送に遅延が生じる直前の論理ブロック・アドレス。

PMI ビットが 0 のときは 2 ～ 5 の論理ブロック・アドレスは 0 とする

図15-1 Read Capacity データ・フォーマット

バイト	内 容
0	最後尾の論理ブロック・アドレス
1	
3	
4	ブロック長(単位バイト) 最後尾の意味は、PMIビットで変化する。
5	
7	
16	CRC エラー検出コード

写真1
ソニーの
CD-ROM 装
置 CDU561



表4 ソニー製 CDU561 シリーズ CD-ROM ドライブの機能仕様(抜粋)

準拠規格: CD-AUDIO (RedBook) および CD-ROM (YellowBook) に準拠

ユーザ・データ記憶領域:	656 M バイト以上(モード1) 748 M バイト以上(モード2)
ユーザ・データ/ブロック:	2048 バイト(モード1, モード2 フォーム1) 2336 バイト(モード2) 2328 バイト(モード2 フォーム2)
対応モード:	CD-DA, CD-ROM(モード1, モード2), CD-ROM XA(モード2 フォーム1 および2), CD-I(モード2 フォーム1 および2), CD-I Ready, CD-Bridge およびフォト CD(シングル, マルチセッション)
ブロック長(単位: バイト):	CD-DA 2448, 2368, 2352 CD-ROM モード1 2352, 2336, 2048, 1024, 512 CD-ROM モード2 2352, 2340, 2336 CD-XA および CD-I フォーム1 2647, 2646, 2352, 2340, 2336, 2056, 2048, 1024, 512 フォーム2 2647, 2646, 2531, 2340, 2336
転送レート:	300 K バイト/秒(モード1, ダブル) 150 K バイト/秒(モード1, ノーマル) 342.2 K バイト/秒(モード2, ダブル) 171.1 K バイト/秒(モード2, ノーマル) SCSI インターフェース 非同期バースト・レート 2.5 M バイト/秒 同期バースト・レート 4 M バイト/秒
アクセス速度:	先頭ブロックへのアクセス終了から最終ブロック・アクセスまで 520 m 秒(ダブル典型値) 550 m 秒(ノーマルの典型値) ブロックからブロックへのランダム・アクセス 295 m 秒(ダブルの典型値) 360 m 秒(ノーマルの典型値)
ユーザ・エラー・レート:	モード1 ECC オン <10 ⁻¹² ブロック/ビット(ダブル) <10 ⁻¹⁵ ブロック/ビット(ノーマル) モード1, 2 ECC オフ <10 ⁻⁹ ブロック/ビット(ダブル) <10 ⁻¹² ブロック/ビット(ノーマル)

6

ソニーのCD-ROMの機能

つぎに実際の CD-ROM ドライブのインターフェース機能を見ます。取り上げたドライブは、ソニーの CDU561 です。いわゆる CD-I からフォト CD までマルチメディアに対応した倍速(ダブル)ドライブです。機能仕様の概略を表4に示しておきます。

■ インターフェース仕様

SCSI ポートは、シングルエンド伝送です。コネクタは 50P ヘッドで、ターミネータがコネクタのすぐ脇にあって着脱できます。デバイス・タイプは、ターゲットです。論理装置については記載がありませんでしたが、シャーシを開けてみたかぎりでは複数ドライブは

意図していないようです(LUN0 のみが見える)。ディスクコネクタ/リコネクタ、コマンド・リンク、同期/非同期転送など、キューイングを除いてスループットを上げるための SCSI 機能はみんな使用できます。

仕様の冒頭に、「SCSI は CD-ROM に特有の命令をすべて備えているわけではないので不足を補う命令が追加されている」と記されています。これは、利用技術が急速に進むデバイスでは致し方のない現象でしょう。CD-ROM のメリットを生かしつつマルチメディアのリアルタイム要求を満たすために、今後もアプリケーション側からの要請で追加命令が増えるかと思われます。

このドライブには標準 SCSI 命令に加えて6種のベンダ・ユニーク命令が加わっています。また、規格にある命令についても一部は機能の拡張がなされています。これらを順に見てゆきましょう。

■ 規格に準拠した命令群

すでに説明した SCSI 標準命令とまったく同形なのが以下の命令群です。これらについては ⑤ を参照してください。もちろんこの他に Inquiry や Read など共通デバイス命令も実装されていますが、説明は省略します。

Pause Resume, Play Audio(10, 12), Play Audio MSF, Play Audio Track Index, Play Audio Track Relative(10, 12), Read Header

■ 拡張機能付きの命令

三つの標準命令についてベンダ・ユニークな機能変更がなされている部分の要点を示します。

▷ Read CD-ROM Capacity 命令

命令表を見るとわかるように PMI ビットと論理ブロック・アドレスがなくなっています(図16)。

▷ Read Sub-Channel 命令

サブチャネルのデータ・フォーマット・コード 00h の扱いが異なっています。

コード値 SCSI X3T9.2/375R 10k SCSI CDU561

00h 全データが送られる 予約

▷ Read TOC 命令

CDU561 には拡張フォーマットがあって SCSI では定義していないマルチセッションのメディアが扱えるようになっています(図17)。

■ ベンダ・ユニーク命令

Audio Scan, Set CD-ROM Speed, Read All Subcodes, Read CD-DA, Read CD-DA MSF, Read CD-XA の 6 種の命令がベンダ・ユニークとして追加されています。これらの命令は単純なオーディオ再生ならば使用されません。

▷ Audio Scan 命令の詳細

指定されたスキャン開始アドレスから前方または、後方にスキャンします。方向は 1 バイト・オフセット 4 ビットの再生方向ビットで決まります(図18)。

▷ Set CD-ROM Speed 命令の詳細

データ・ブロックにアクセスする際のスピンドルの回転数を与えます。デフォルトはダブルです。この命令は変わり者で、SCSI-1 の LUN フィールドが定義されていません(図19)。

▷ Read All Subcodes 命令の詳細

ターゲットがオーディオ再生中に読み取っているサ

図16(a) Read CD-ROM Capacity 命令 X3T9.2/375R 10k

ビット バイト	7	6	5	4	3	2	1	0	
0	OP コード (25h)						RelAdr		
1	LUN			予約					
2	PMI 論理ブロック番号								
3									
4									
5									
6	予約								
7									
8	予約						PMI		
9	コントロール								

図16(b) Read CD-ROM Capacity 命令 CDU561

ビット バイト	7	6	5	4	3	2	1	0
0	OPコード(25h)							
1	LUN			予約				予約
2	予約							
3								
4								
5								
6	予約							
7								
8	予約						予約	
9	コントロール							

ブコード(P から W チャンネルの全データ)をホストへ送ります(図20)。

▷ Read CD-DA 命令の詳細

指定した論理ブロック・アドレスからのオーディオ・データとサブ Q コードをホストへ送ります。オーディオ・データは、普通 MSF アドレスで扱われますが、本命令では論理ブロックでの指定になります。MSF アドレスからそれに等しい論理ブロックを算出するには以下の計算を行います。

論理ブロック・アドレス

$$= (M - Ms) \times 60 \times 75 + (S - Ss) \times 75 + (F - Fs)$$

ここで M, S, F はホストがほしいデータの先頭 MSF アドレスです。Ms, Ss, Fs はメディアの第 1 トラックの第 1 ブロックに相当する MSF アドレスです。この Ms, Ss, Fs は Read TOC 命令によって知ることができます(図21)。

図17(a) Read TOC 命令 X3T9.2/375R 10k

ビット バイト	7	6	5	4	3	2	1	0
0	OP コード (43h)							
1	LUN			予約			MSF	予約
2	予約							
3								
4								
5								
6	開始トラック							
7	転送長							
8								
9	ベンダ定義		コントロール					

図17(b) Read TOC 命令 CDU561

バイト \ ビット	7	6	5	4	3	2	1	0
0	OP コード (43 h)							
1	LUN			予約			MSF	予約
2	予約							
3								
4								
5								
6	開始トラック/セッション番号							
7	転送長							
8								
9	フォーマット	コントロール						

値	内 容
00b	SCSI-2 規格と互換な TOC データを送る
01b	最初のセッション番号、最後のセッション番号、最後のセッションの開始アドレスを送る
10b	指定されたセッション番号からの TOC 内にあるすべての Q サブコードを送る。

●フォーマット・コード 10b の場合の TOC データ・フォーマット

バイト	内 容
0-1	TOC データ長
2	先頭セッション番号
3	末尾セッション番号
4以降	トラック記述域

●トラック記述域

バイト	内 容
0	セッション番号
1.[7-4]	ADR
1.[3-0]	コントロール
2	バイト 1/TNO
3	バイト 2/ポイント
4	バイト 3/分
5	バイト 4/秒
6	バイト 5/フレーム
7	バイト 6/ゼロ
8	バイト 7/P 分
9	バイト 8/P 秒
10	バイト 9/P フレーム

図18 Audio Scan 命令

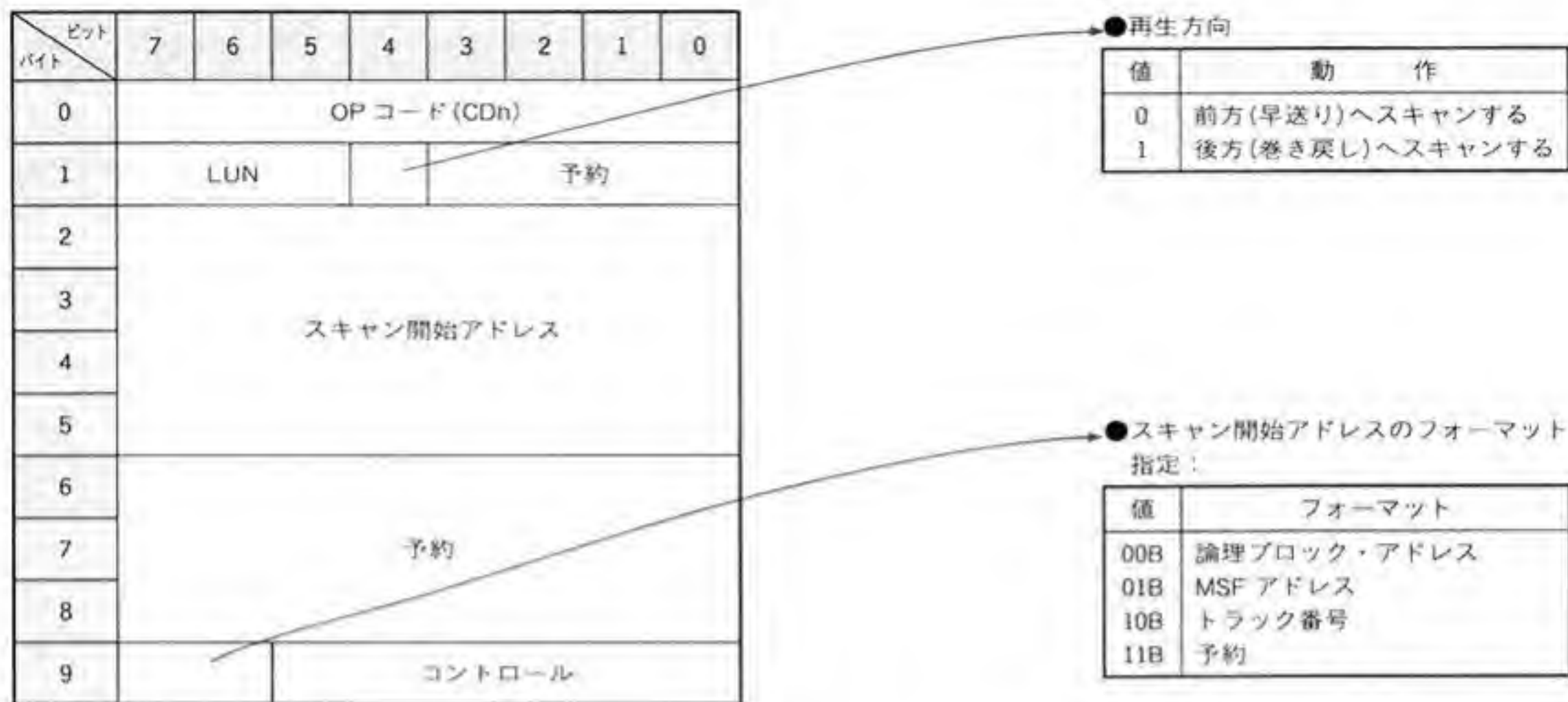


図19 Set CD-ROM Speed 命令

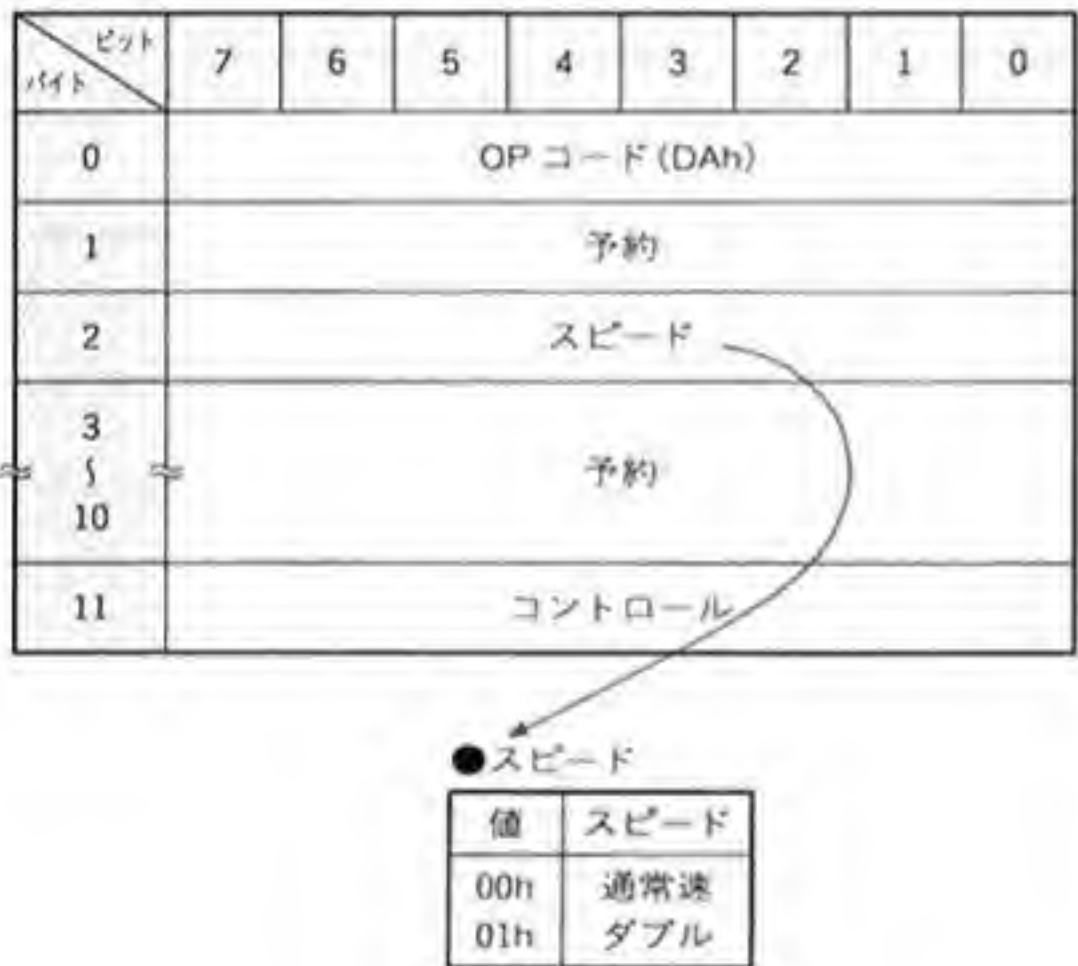


図20 Read All Subcodes 命令



以下にサブコード 01h の場合の転送ブロック・フォーマットを示します。ホストはバイト 0 を最初に受信した後 1, 2, 3 と 2367 まで受信します。

バイト	内容
0-2351	オーディオ・データ 4 バイトのくり返し
2352.[7-4]	制御データ 4 ビット
2352.[3-0]	ADR データ 4 ビット
2353	トラック番号 (BCD)
2354	インデックス番号 (BCD)

2355	M トラック相対分 (BCD)
2356	S トラック相対秒 (BCD)
2357	F トラック相対フレーム (BCD)
2358	予約 (00h)
2359	M 絶対分 (BCD)
2360	S 絶対秒 (BCD)
2361	F 絶対フレーム (BCD)
2362-2367	00h

オーディオ・データ 4 バイトのくり返し: 「左チャネル下位 8 ビット→左チャネル上位 8 ビット→右チャネル下位 8 ビット→右チャネル上位 8 ビット」

図21 Read CD-DA 命令

ビット バイト	7	6	5	4	3	2	1	0
0	OP コード (D8h)							
1	LUN			予約				
2	転送の先頭ブロック・アドレス							
3								
4								
5								
6	転送長							
7								
8								
9								
10	サブコード							
11	コントロール							

●転送長：転送するブロック数だが、1ブロック当たりのバイト数はサブコードの値で決まるので、全転送バイト数はサブコードと転送長に依存する。

●サブコード：

値	ブロック長 (単位バイト)	説明
00h	2352	サブコードなしの CD-DA データのみ
01h	2368	サブ Q コード付きの CD-DA データ
02h	2448	すべてのサブコード付き の CD-DA データ
03h	96	すべてのサブコードのみ
04-FFh		予約

ネル下位 8 ビット→右チャネル下位 8 ビット」このくり返しとなります。

▷ Read CD-DA MSF 命令の詳細

Read CD-DA 命令と同様のオーディオ・データをホストへ送ります。本命令では転送データの区間指定を MSF アドレスで行います(図22)。

▷ Read CD-XA 命令の詳細

指定した論理ブロック・アドレスからの CD-XA データをホストへ送ります(図23)。

図22 Read CD-DA MSF 命令

ビット バイト	7	6	5	4	3	2	1	0
0	OP コード (D9h)							
1	LUN			予約				
2	予約							
3	転送開始 M フィールド							
4	転送開始 S フィールド							
5	転送開始 F フィールド							
6	予約							
7	転送終了 M フィールド							
8	転送終了 S フィールド							
9	転送終了 F フィールド							
10	サブコード							
11	コントロール							

7

CDドライブの動作

Macintosh に CDU561 を接続して、システム立ち上げから音楽 CD を再生するまでの SCSI バス動作をモニタ OZ202 で観測した結果を図24 に示します(CD-ROM 以外は削除するなどデータは編集してある)。この実験ではメディアはロード済みにして、CDU561 の SCSI-ID は 3 にしました。また、ソフトは FWB の CD-ROM ToolKit を使用しました。

図23 Read CD-XA 命令

ビット バイト	7	6	5	4	3	2	1	0
0	OP コード (DBh)							
1	LUN			予約				
2	転送の先頭論理ブロック・アドレス							
3								
4								
5								
6	転送長							
7								
8								
9								
10	CD-XA フォーマット							
11	コントロール							

● 転送の先頭論理ブロック・アドレス：
つぎの計算式にしたがう。
先頭論理ブロックアドレス = $(M-0) \times 60 \times 75 + (S-2) \times 75 + (F-0)$

● CD-XA フォーマット：

値	ブロック長 (単位バイト)	説 明
00h	2048	ユーザ・データのみ
01h	2352	CD-XA のすべてのセクタ・データ
02h	2646	CD-XA のすべてのセクタ・データと 294 バイトのエラー・フラグ
03-FFh		予約

動作の概略はつぎのようになります。

- (1) SCSI リセットが発行される
- (2) Read 命令でブロック 0 を読もうとするが、CD-DA なのでチェック・コンディションとなる
- (3) Inquiry 命令でデバイスの型を知る
- (4) Read CD-ROM Capacity 命令でメディアの容量を知る
- (5) Prevent Allow Medium 命令でメディアをロックする
- (6) Test Unit Ready 命令でユニットの状態を知る
- (7) Mode Sense と Mode Select 命令でモードを決定する
- (8) ブロック 12h に対し Read Header 命令を発行する
- (9) Read TOC を複数のフォーマット・コードで実行する
以上で立ち上げが終了します。
- (10) オーディオ・コントローラのアイコンを画面でクリックすると、Mode Sense と Read Sub-Channel をくり返し、メディアの状態をスクリーンに表示する

- (11) オーディオ・コントローラから再生命令を送ると Play Audio Track/Index 命令が発行される
- (12) スピーカから音楽を流している間、Mode Sense と Read Sub-Channel が繰り返されて演奏時間などがオーディオ・コントローラに表示される

なお、本稿の執筆にあたって、ソニー(株)の中野氏のご協力をいただきました。誌面を借りてお礼申し上げます。

参考文献

- 1) CD-ROM DRIVE SCSI INTERFACE MANUAL MODEL CDU561, SONY Corp.
- 2) Product Specification for CDU561 CD-ROM Drive, Rev 1.0 SONY Corp.
- 3) John Donovan, "EDN-Technology Feature", EDN JUL 22, 1993
- 4) 安田直義, 「CD-ROM は素敵」, 「UNIX マガジン」 1992.12~1993.12
- 5) ANSI X3T9.2 WORKING DRAFT 375R Revision 10k 17-MAR-93, ANSI

ふたがみ・たかお 前出

図24 MacにCDU561を接続してSCSIモニタで観測した結果は…

Mac II+FWB CD-ROM Toolkit → SONY CDU-561 SCSI-ID3											
FREE for			BUSY for			State	Nexus	Information Transfer phases			
s: ms. us. ns	s: ms. us. ns	s: ms. us. ns	Memory	ADRS	ITLRD	Msg Out	Command	Data	Status	Msg In	
***** Reset Condition. RST pulse width = 26.250 [us] *****											
2.436.239.850	.256.787.600	T	0	1	76	C	***** Selection Timeout. ID = 6. Period = 256.783 [ms]				*****
.85.950	.256.787.600	T	6	75	C	***** Selection Timeout. ID = 5. Period = 256.783 [ms]					*****
.81.200	.256.787.600	T	11	74	C	***** Selection Timeout. ID = 4. Period = 256.783 [ms]					*****
.83.450	.3.336.750	T	16	730	C		READ (LBA=000000H)		CHECK CND	CMD CMPLT	
.19.800	.3.347.600	T	29	730	C		READ (LBA=000000H)		CHECK CND	CMD CMPLT	
.72.800	.256.787.600	T	42	72	C	***** Selection Timeout. ID = 2. Period = 256.783 [ms]					*****
.79.850	.256.787.600	T	47	71	C	***** Selection Timeout. ID = 1. Period = 256.783 [ms]					*****
.312.400	.753.800		2642	70	C						
.208.000	.256.787.650		2647	71	C	***** Selection Timeout. ID = 1. Period = 256.783 [ms]					*****
.105.400	.256.787.600		2652	72	C	***** Selection Timeout. ID = 2. Period = 256.783 [ms]					*****
.85.150	.3.923.500		2657	730	C		INQUIRY	36	GOOD	CMD CMPLT	
.186.950	.256.787.650		2706	74	C	***** Selection Timeout. ID = 4. Period = 256.783 [ms]					*****
.92.500	.256.787.650		2711	75	C	***** Selection Timeout. ID = 5. Period = 256.783 [ms]					*****
.88.950	.256.787.650		2716	76	C	***** Selection Timeout. ID = 6. Period = 256.783 [ms]					*****
.7.559.700	.3.803.500		2771	730	C		INQUIRY	I	36	GOOD	CMD CMPLT
.5.325.100	.22.030.200		2845	70	C						
.4.426.050	.5.906.800		2850	730	C		INQUIRY	I	36	GOOD	CMD CMPLT
.173.200	.4.356.400		2899	730	C		INQUIRY	I	36	GOOD	CMD CMPLT
.444.550	.3.101.950		2948	730	C		READ CDROM CAPACITY	I	8	GOOD	CMD CMPLT
.180.150	.3.072.850		2973	730	C		READ CDROM CAPACITY	I	8	GOOD	CMD CMPLT
.112.150	.3.993.250		2998	730	C		MODE SENSE	I	20	GOOD	CMD CMPLT
.87.150	.20.983.650		3031	730	C		MODE SELECT	O	20	GOOD	CMD CMPLT
.64.950	.3.336.700		3064	730	C		READ CDROM CAPACITY	I	8	GOOD	CMD CMPLT
.97.250	.3.347.100		3089	730	C		READ CDROM CAPACITY	I	8	GOOD	CMD CMPLT
.22.586.050	.6.158.200		4284	730	C		PREVENT ALLOW MEDIUM		GOOD	CMD CMPLT	
.16.228.750	.3.738.600		5162	730	C		TEST UNIT READY		GOOD	CMD CMPLT	
.94.500	.3.621.900		5175	730	C		READ CDROM CAPACITY	I	8	GOOD	CMD CMPLT
.102.600	.4.179.650		5200	730	C		PREVENT ALLOW MEDIUM		GOOD	CMD CMPLT	
.70.800	.3.371.050		5213	730	C		READ CDROM CAPACITY	I	8	GOOD	CMD CMPLT
.103.850	.4.004.300		5238	730	C		MODE SENSE	I	20	GOOD	CMD CMPLT
.89.200	.20.679.300		5271	730	C		MODE SELECT	O	20	GOOD	CMD CMPLT
.67.300	.3.457.000		5304	730	C		READ CDROM CAPACITY	I	8	GOOD	CMD CMPLT
.116.350	.348.624.200		5329	730	C		READ HEADER		CHECK CND	CMD CMPLT	
.76.050	.4.122.050		5346	730	C		REQUEST SENSE	I	16	GOOD	CMD CMPLT
.173.600	.3.201.550		5375	730	C		READ CDROM CAPACITY	I	8	GOOD	CMD CMPLT
.29.288.100	.13.358.150		5400	70	C						
.1.713.700	.5.598.600		5430	730	C		READ TOC	I	12	GOOD	CMD CMPLT
.490.050	.6.539.900		5459	70	C						
.1.375.200	.4.663.000		5464	730	C		PREVENT ALLOW MEDIUM		GOOD	CMD CMPLT	
.84.450	.3.904.500		5477	730	C		PREVENT ALLOW MEDIUM		GOOD	CMD CMPLT	
.6.441.150	.5.648.450		5490	730	C		READ TOC	I	20	GOOD	CMD CMPLT
.376.500	.4.800.400		5527	730	C		READ TOC	I	12	GOOD	CMD CMPLT
.1.067.750	.14.455.400		5556	730	C		READ TOC	I	76	GOOD	CMD CMPLT
.299.050	.4.391.750		250	730	C		MODE SENSE	I	28	GOOD	CMD CMPLT
.311.800	.4.946.400		291	730	C		MODE SENSE	I	28	GOOD	CMD CMPLT
.86.300	.28.587.550		332	730	C		MODE SELECT	O	28	GOOD	CMD CMPLT
.410.039.100	.10.346.300		686	730	C		READ SUB-CHANNEL	I	16	GOOD	CMD CMPLT
.223.398.650	.10.738.850		719	730	C		READ SUB-CHANNEL	I	16	GOOD	CMD CMPLT
.67.950	.13.190.650		752	730	C		READ SUB-CHANNEL	I	16	GOOD	CMD CMPLT
.503.150	.12.484.150		785	730	C		READ SUB-CHANNEL	I	16	GOOD	CMD CMPLT
.69.350	.12.332.900		818	730	C		READ SUB-CHANNEL	I	16	GOOD	CMD CMPLT
.466.700	.4.463.450		851	730	C		MODE SENSE	I	28	GOOD	CMD CMPLT
.117.450	.692.872.500		892	730	C		PLAY AUDIO TRACK IND		GOOD	CMD CMPLT	
.2.948.400	.9.271.700		909	730	C		READ SUB-CHANNEL	I	16	GOOD	CMD CMPLT
.403.581.800	.4.859.200		942	730	C		MODE SENSE	I	28	GOOD	CMD CMPLT
.412.231.450	.10.650.300		983	730	C		READ SUB-CHANNEL	I	16	GOOD	CMD CMPLT
.2.742.900	.10.666.350		1016	730	C		READ SUB-CHANNEL	I	16	GOOD	CMD CMPLT

オープン・システム入門教室

Unix, LAN, WAN, RDBMS…広い領域にわたる技術を簡潔に整理した待望の書!

ダウンサイジングに対処するための10科目 小暮裕明著 1,650円(税込)

CQ出版社

■ スキャナの一般論

スキャナとは、一般的に空間分布した情報を逐次的にもれなくとらえて伝えるための装置のことです。その適用範囲はじつに広大です。生まれてくる前から子供の性別がわかってしまう超音波スキャナ、原子の結晶構造すら写し出せる原子間力顕微鏡など、原理的な分類ではどれもスキャナ的一种です。熱、圧力、密度、速度など空間分布しているものは、すべてスキャンの対象になりうるわけですから、この範囲の広さは当然といえば当然かもしれません。

SCSI で定義しているスキャナは、ここまで広い概念ではありません。スキャナの章にある概説とスキャナ命令一覧をみるとわかるように、SCSI 規格でのスキャナとは、光学的に走査可能な 2 ないし 3 次元物体のデジタル表現を生成する装置を指します。すなわち、印刷物、写真、絵など長方形の紙やフィルムにのった画情報を逐次的に取り込んで電子情報のブロックにする装置を意味するわけです。

SCSI のスキャナ規定は、実際にスキャナ・アプリを作成するうえでは決して必要十分な規定ではありません。しかし、最低限ここまで決めておけば、各社各様なインターフェースでソフトの互換性をとるのに四苦八苦するよりははるかに楽になります。スキャナのメーカーとしては特定のソフトに依存しないで製品の販売ができるのですから、より広いマーケットで活動ができることになります。

現時点ではまだこのようなレベルにいたっていませんが、基本機能を共通のインターフェースでという流れは明らかです。マイクロソフトの AtWork 構想に対して日本の事務機メーカーは非常に積極的で、相互の協力関係も進んでいるようです。このような複雑で大規模なシステム構想を実現するときに内部機器間のインターフェースが個々ばらばらで、それぞれに複雑な I/O 手続きが必要であって 100 年あっても目標に到達しないでしょう。こうした観点でみると SCSI は、スキャナやプリンタなど事務機のデータ交換と統合化にも内部バスとして大きな意味をもつインターフェースです。

1

スキャナの基本諸元と SCSI の関係

本稿では、スキャナ本体の解説が目的ではありませんから、内部的な機構にはふれずに、事務用品としてスキャナを使う立場から見た機能が SCSI によるインターフェース制御とどのような関係にあるかを考えます。

● 最大読み取り領域

光学センサがカバーできる矩形の範囲のことです。図 25 のように主走査方向を X 軸、副走査方向(メカ的にセンサが動く方向)を Y 軸として $X \times Y$ で示します。この最大値自体はセンサやセンサの移動範囲など機構的制限によるものですから、制御の対象ではありません。しかし、A4 版のスキャナで葉書のデータを読もうと思ったら読み取り範囲を限定したほうが時間短縮であると同時にデータ・サイズも小さくなります。このように読み取り領域の部分集合を定義してスキャンを限定する方法は SCSI でウィンドウとして定義されています。

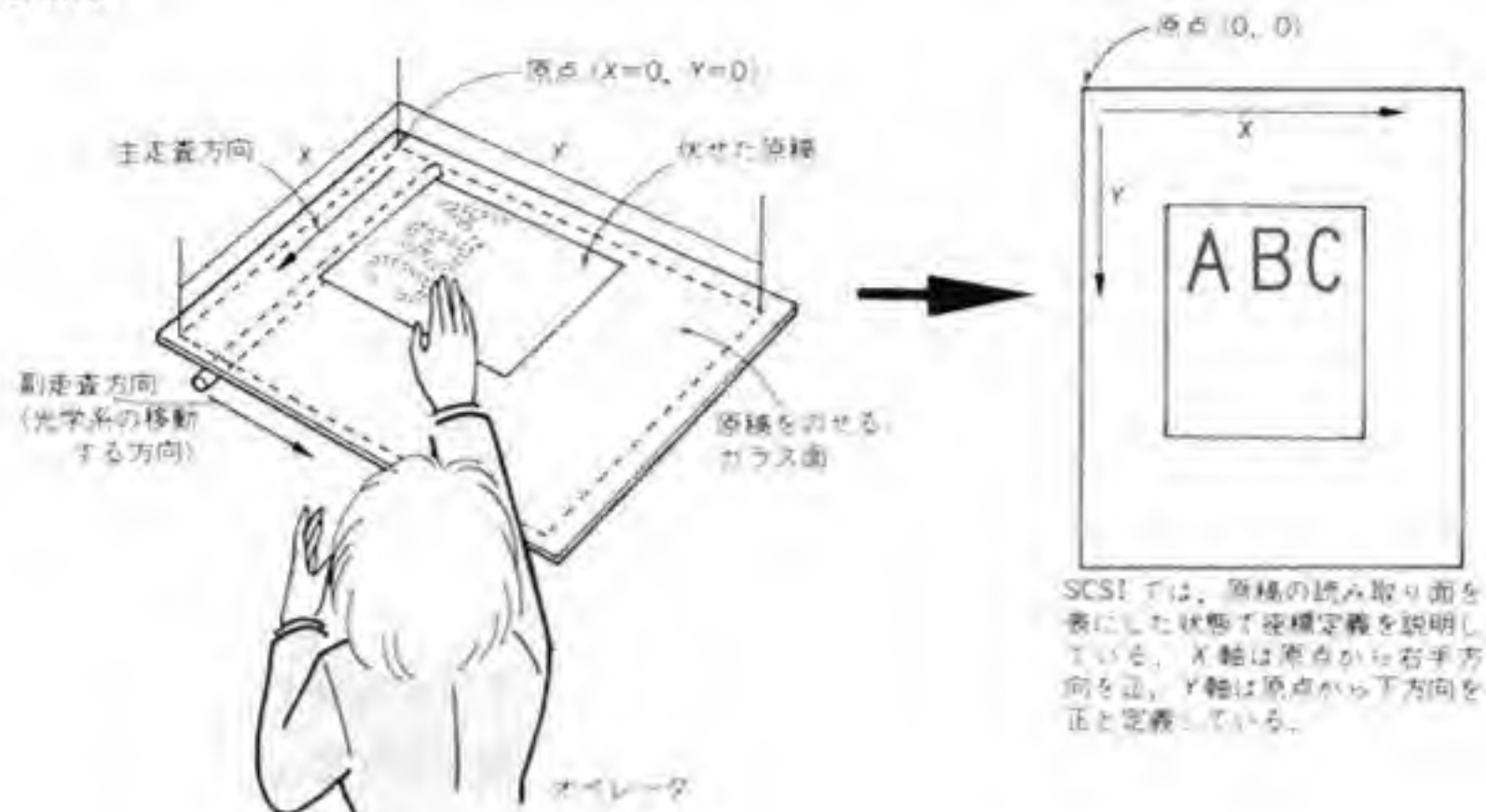
● 読み取り分解能

1 インチ当たりのイメージ読み取りピッチです。もちろん、この数値が大きいほどきめの細かな絵がデータとして取り込めることになります。現状では原稿読み取りならば 300 から 400 DPI (Dot Per Inch) 程度が使われます。また、事務用写真スキャナでは 1200 から 2400 DPI です。ちなみに 400 DPI は、1 mm につき 16 点弱のデータ取り込みに相当します。分解能の具体的な値は、SCSI に定義されていません。もちろん、スキャナ・メーカーは、この値を定めています(後述の IS410 の例を参照)。

● 読み取り時間

所定の読み取り領域(通常 A4 版)をスキャンするのに要する時間です。つまり、どう工夫してもこれより

図25 最大読み取り領域



早く原稿を読み取ってホストに転送することは不可能な限界時間のことです。これに SCSI のスループットを加味すれば実質の読み取り時間となります。SCSI のスループットは、信号方式、ハンドシェイク、バス幅、バッファ方式とサイズなど SCSI の基本技術とホストの能力をどう利用するかにかかっています。事務機としての現在のスキャナであれば、シングルエンド 8 ビットの非同期転送で十分でしょう。

現存するスキャナ・インターフェースはこのベーシックな方式が主流です。しかし SCSI を共有バスととらえる場合には、画像データのような大容量データを流されてバスを何秒も占拠されては他の通信の妨げになります。よって今後はスキャナ側がサポートする SCSI プロトコルも、より高度なものになると思われます。

● データ種類

読み取りできるデータの種類の種類です。文字データであればデータ種類を 2 値(白黒)として原稿を読めば十分ですが、絵や写真では疑似中間調表現(ハーフトーンともいう)または多値として読まないで中間調の再現ができません。これらは SCSI で共通の設定方式が定められています。

● データ処理

圧縮と加工と認識の 3 系統のデータ処理があります。圧縮は冗長なデータをより少ないビット数で表現することで記憶に要するメモリを節約したり、ホストへの

データ転送時間の短縮をはかります。

圧縮アルゴリズムについては規格による取り決めはないので、CCITT への参照が示されています。加工は、スキャナのセンス特性(反射率と濃度が線形に対応していない)を補正します。認識は輪郭を強調したり黒いしみを除くなど画像としての強調処理を行います。これらもアルゴリズムについてはまったく言及されていません。

● データ・フォーマット

Set Window のパラメータ・フォーマットの一部分が規定されていますが、画像データの内容にかかわる規定はありません。事例として後述の IS410 を見てください。

2

SCSIで標準化されている スキャナ機能

X3T9.2 の 15 章で示されているスキャナ・インターフェース仕様で定義されているスキャナの機能がサポートされたスキャナにどのようなことができるかを簡単なシナリオで紹介しましょう。

■ スキャナの必須命令

スキャナの機能はターゲットとして実装されます。初めにスキャナとしてサポートが必須とされている

SCSI 命令の概略を示します。

● Test Unit Ready と Request Sense 命令

論理装置が利用可能かどうかを調べるもっとも基本的な命令です。スキャナはハード・ディスクなどと違って、ユーザが手で触れて操作する機構部分があります。電子的な装置状態に加えてこれらの機構が適切な位置にない場合にはスキャナは本命令に対してチェック・コンディションを返します。たとえば、蓋が少し開いているなど、スキャナが動作するためにオペレータの介入が必要な機器状態に対してチェック・コンディションとなります。イニシエータが本命令を発行してチェック・コンディションとなったときには、Request Sense 命令で装置状態を調べることになります。これも定石のとおりです。

● Inquiry 命令

装置の固有情報をイニシエータへ返します。全デバイス共通情報に加えて、スキャンできる領域の寸法など、スキャナの基本諸元が取得できます。ホストが、多種類のスキャナを使えるためには、この命令でスキャナの固有情報を読み出して対応する必要があります。

● Send Diagnostics 命令

標準的な SCSI 命令です。スキャナに特有なものはありません。

● Reserve Unit, Release Unit 命令

標準的な SCSI 命令です。スキャナを特定のホストが独占したり独占を解除するために使います。原稿の読み取りは特定のホスト・マシンで動いているアプリに強く結びつく性質の仕事ですから、複数イニシエータでの応用にあたってはスキャナを占有してから使用するのが正当な方法になります。

● Set Window 命令

スキャナに特有な概念であるウィンドウを定義します。ウィンドウを定義することでスキャン可能領域の内部の1区画を切り出して個々のウィンドウごとに独立した操作対象として扱えるようになります。

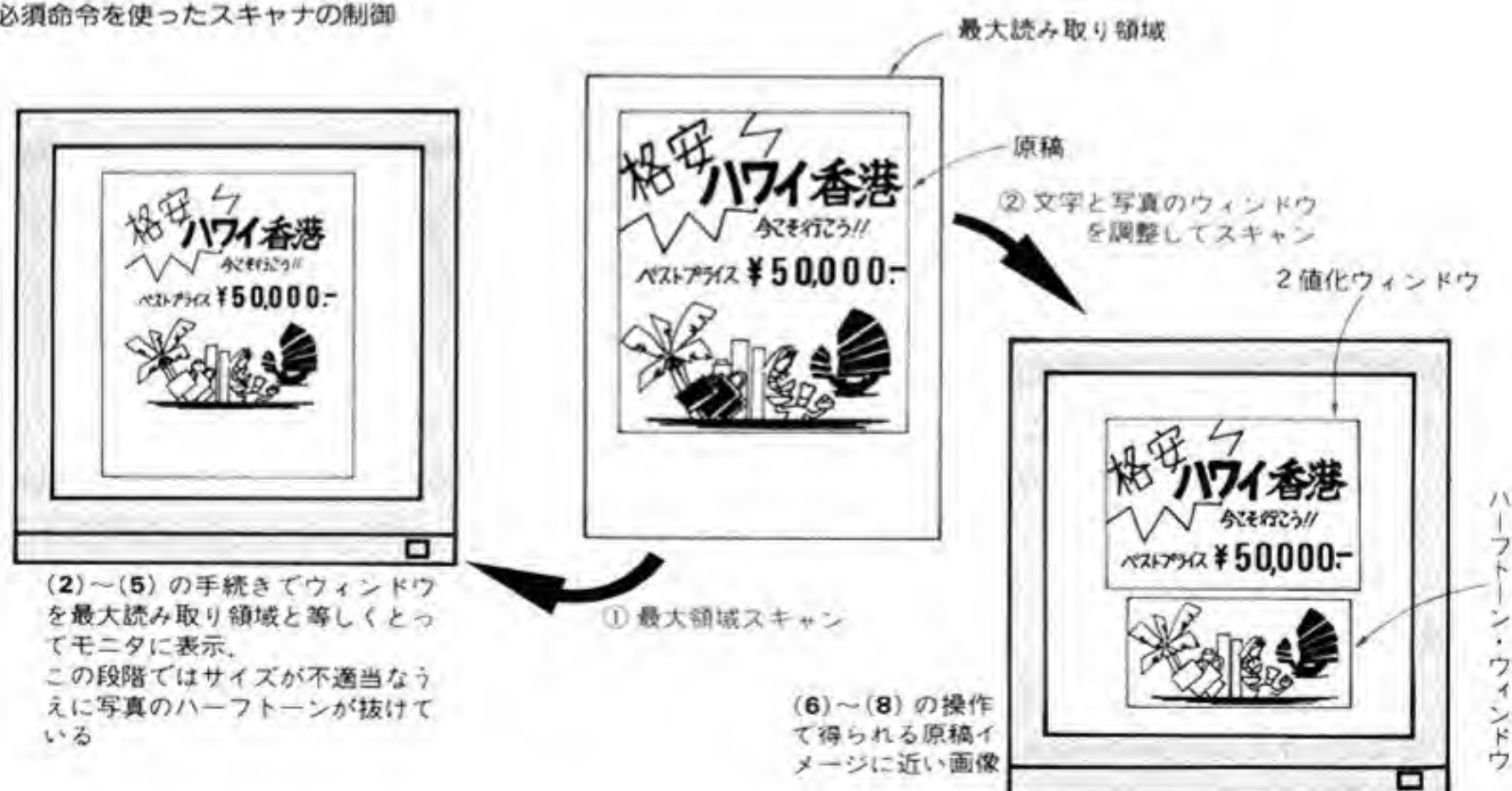
● Read 命令

スキャンした画像を初めとするスキャナのデータをホストへ転送します。後述の例では、Set Window してしまえば、Window を指定して Read 命令をスキャナに送るだけで、スキャンを実行したうえで画像をホストへ送るという複合動作をしてくれます。

● Scan 命令

必須ではありませんが、重要な命令です。Scan 命令を受けるとスキャナは画像の走査を開始します。

図26 必須命令を使ったスキャナの制御



■ 必須命令を使ったスキヤナの制御

すでに原稿がセットされているスキヤナに対してホスト(イニシエータ)がどのような操作を行って2値画像を取得するのかわ、仮想のシナリオを作って説明しましょう。原稿には、写真と文章が混在していると想定します。また、スキヤナは、SCSIの規格に準拠している仮想的なものを使うと考えます。さらに、ここではディスクコネクタなどは除いて基本的なプロトコルだけを使います(図26)。

(1) Test Unit Ready 命令を発行します。まず、スキヤナ・ユニットが電子的、機械的にスキャン可能であるかどうかを調べます。応答が Good ならばつぎのステップへ進みます。

(2) Inquiry 命令でスキャンの最大領域や機能を知ります。この情報をもとにして Set Window のパラメータを算出します。もし、スキヤナのパラメータが既知であればこの命令は不要です。

(3) Set Window 命令でウィンドウを定義します。とりあえず最大読み取り領域を全部スキャンできるようなウィンドウを一つセットします。ウィンドウのデータ特性は、白黒の2値とします。

(4) Scan 命令で原稿を実際にスキャンします(Scan 命令は必須ではない。ここで Scan 命令を送らない場合はつぎのステップの Read を実行すると、Scan 動作→データ送出になる)。スキャンした結果のデータはバッファに貯えられます。

(5) Read 命令でバッファにある画像データをホストへ転送します。

画像データのフォーマットの詳細は、規格に定められていないのでスキヤナの仕様書などからその情報を得ておく必要があります。ホストは、リードした画像データをスクリーンに表示します。オペレータはスキャン領域のトリミング(原稿が小さいとき)や、写真のある部分域の指定を画面で行います(リード後の作業はスキヤナとは独立なホストでの作業ですから SCSI とは関係ない)。

(6) Set Window 命令を再度発行して、前項で求められた新領域と特定写真領域をあらためてウィンドウとして登録します。写真領域は、ハーフトーン2値化などの処理をウィンドウ・パラメータとして指定します。

(7) Scan 命令をもう一度発行します。この Scan では

表5 ページ・コード F0h のデータのスキヤナ固有情報

オフセット	内 容
2	JIS バージョン
5-6	等倍読み取り時の X 軸方向の解像度(単位 DPI)
7-8	等倍読み取り時の Y 軸方向の解像度(単位 DPI)
9. [7-4] *	X 軸方向の解像度の可変刻み幅(単位 DPI)
9. [3-0] *	Y 軸方向の解像度の可変刻み幅(単位 DPI)
10-11	X 軸方向の論理的な解像度の最大値(単位 DPI)
12-13	Y 軸方向の論理的な解像度の最大値(単位 DPI)
14-15	X 軸方向の論理的な解像度の最小値(単位 DPI)
16-17	Y 軸方向の論理的な解像度の最小値(単位 DPI)
18	利用可能分解能マップ-1
19	利用可能分解能マップ-2
20-23	ウィンドウ幅
24-27	ウィンドウ長さ MSB
28	実装機能マップ

*たとえば9. [7-4] は9バイト目の7-4ビットのフィールドを表す

ウィンドウ・リストに前項の新ウィンドウを指定します((4)と同様に Scan 命令は必須ではない)。

(8) Read で領域を読み出します。これをホストの画面に表示すれば、きれいにトリミングされたうえに写真部分はハーフトーンで表現された原稿のイメージがデジタル化されたことになります。

以上は仮想 SCSI スキヤナの動作でした。実際には Set Window の前に Mode Select などの設定命令が必要とされる場合があります。また、複数枚の原稿を連続読み取りするために原稿送り装置(ドキュメント・フィーダとかローダと呼ばれる)の設定をする場合もあります。この制御は必須命令を受けたスキヤナが自動的に実行、またはオプションとして Object Position 命令で実現されます。

この例でわかるとおり、スキヤナにとって重要なパラメータは、Inquiry と Set Window および Mode Select で設定されます。そして機器動作としては Scan 命令と Object Position 命令に特徴があります。

以下、これらのパラメータと命令について説明します。

■ Inquiry 命令の詳細

イニシエータが SCSI を経由してスキヤナの仕様を読み出すには、つぎのようなシーケンスで情報を取り出します。

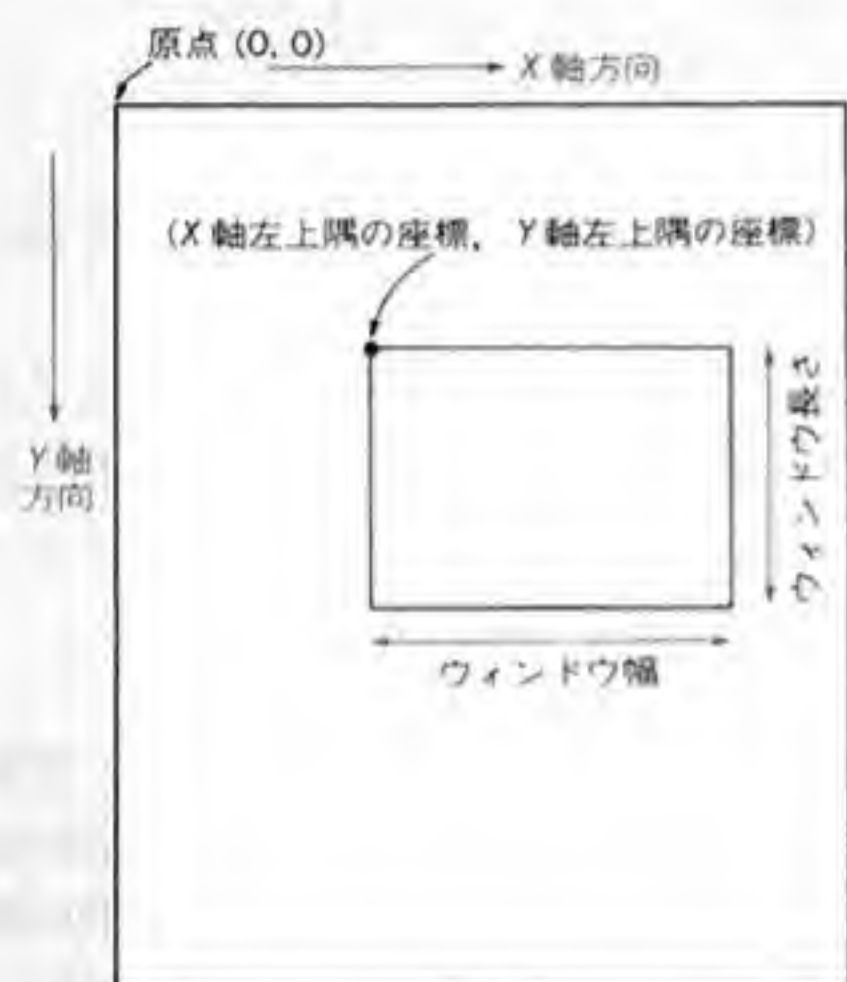
- ・ Inquiry 命令を EVPD ビット=0、ページ・コード=0 とセットして発行する。製造者コードなど標準デ

ータを受信する。

- Inquiry 命令を EVPD ビット=1, ページ・コード=0 とセットして発行する。スキャナが保持しているページ・コード・リストを受け取る。
- ページ・コード・リストにあるページ・コードをつけながら Inquiry を繰り返し、すべてのデータを受け取る。

このときスキャナの製品によっては、メーカーで規定したページ・コードによって固有情報を受け取ることが可能です。これを見れば最大読み取り領域や読み取

図27 読み取り領域とウィンドウ



り分解能、オプション機能などのスキャナ仕様がオンラインで読み出せます(表 5 参照)。

■ Set Window 命令の詳細

図27 に読み取り領域とその内部に定義するウィンドウとの位置関係を示します。ウィンドウには、このような幾何的な定義に加えて、その範囲の中での画像読み取り条件やデータ加工の方法までも一緒に定義します。

Set Window 命令(図28)自体は、きわめて簡単な命令です。ウィンドウにかかわる定義は、命令に続いてホストから渡されるデータに含まれています。これらデータの詳細は、ウィンドウ記述域で説明されています(図29)。

■ ウィンドウ記述域の項目説明

ウィンドウ識別子：複数のウィンドウに固有な番号で後々 Scan 命令などで、どのウィンドウをスキャンするかをホストが指示するときに使用します。

オート指定：このビットが1の場合にはスキャナがここで指定されているウィンドウの中にさらにサブウィンドウを自動生成することを許します。ビットが0の場合はそれを許しません。生成されたサブウィンドウのパラメータは Get Window 命令で読むことができます。

X 軸分解能：主走査方向(光学的、電子的に走査する方向)の DPI 単位での読み取り分解能です。0 はデフォ

図28 Set Window 命令とデータ・ヘッダ

ビット バイト	7	6	5	4	3	2	1	0
0	24h(OPコード)							
1	LUN			予約				
2	予約							
3								
4								
5								
6	転送データ長							
7								
8								
9	コントロール							

(a) Set Window 命令

ビット バイト	7	6	5	4	3	2	1	0
0	予約							
1								
2								
3								
4								
5								
6	ウィンドウ・ディスクリプタ長 (ウィンドウ一つ当たりの記述長：単位バイト数)							
7								

(b) データ・ヘッダ (Set Window 命令に続いて出力される)

ルト指定を意味します。

Y 軸分解能：副走査方向(モータで光学センサを移動する方向)の DPI 単位での読み取り分解能です。0 はデフォルト指定を意味します。

左上隅の X 座標：ウィンドウの左上隅が位置する読み取り領域での X 座標をユーザの指定した単位で与えます。なお、ユーザ単位については後述の Mode Select 命令を参照してください。

左上隅の Y 座標：ウィンドウの左上隅が位置する読み取り領域での Y 座標をユーザの指定した単位で与えます。

ウィンドウ幅：ウィンドウの幅をユーザの指定した単位で与えます。

ウィンドウ長：ウィンドウの長さをユーザの指定した単位で与えます。

明るさ：スキャンをするときの光源の明るさを与えます。0 はデフォルト値もしくは実装されているならば自動輝度コントロールを指定します。0 以外の値は相対輝度値を意味し、255 が最高値、1 が最低値、128 が典型値です。

2 値変換のしきい値：スキャンしたデータを白黒の 2 値に変換する場合、白とするか黒とするかを切り分ける値を指定します。0 はデフォルト、もしくは実装されていれば自動しきい値コントロールを指定します。

コントラスト：スキャンしたデータのコントラスト補正の値を指示します。

画像変換方法：ここで指定したコードによってスキャンしたデータをどのような出力データとするかを指定します。

コード	変換方法
00h	2 値白黒
01h	ディザ・ハーフトーン白黒
02h	多階調白黒(グレー・スケール)
03h	2 値 RGB カラー
04h	ディザ・ハーフトーン RGB カラー
05h	多階調 RGB カラー
06~FFh	予約

画素当たりのビット数：単色の明度を表現するのに使うビット数を与えます。

RIF 反転画像指定：2 値の画像変換について、RIF=0 ならば白は 0、黒は 1 とみなされます。RIF=1 では、この逆となります。

パディング方法：変換したデータをホストへ送るときには、1 バイトに複数画素のデータをつめて送ります。

図29 ウィンドウ記述域(図28b)のデータ・ヘッダに続いて出力される)

ビット バイト	7	6	5	4	3	2	1	0
0	ウィンドウ識別子							
1	予約							オート指定
2	X 軸分解能(単位 DPI)							
3								
4	Y 軸分解能(単位 DPI)							
5								
6 }	X 軸左上隅の座標(ユーザ指定単位)							
9								
10 }	Y 軸左上隅の座標(ユーザ指定単位)							
13								
14 }	ウィンドウ幅(ユーザ指定単位)							
17								
18 }	ウィンドウ長(ユーザ指定単位)							
21								
22	明るさ							
23	2 値化変換のしきい値							
24	コントラスト							
25	画像変換方法							
26	画素当たりのビット数							
27	中間調パターン							
28								
29	予約				パディング方法			
30	ビット順列指定							
31								
32	圧縮コード							
33	圧縮引き数							
34 }	予約							
39								
40 }	ベンダ・ユニーク							
47								

RIF
反転画像
指定

しかし、ウィンドウのサイズから端のデータによっては1バイトに空きができます。この空きをうめる方法をコードで指定します。

コード	方 法
00h	空きをうめない
01h	バイト境界まで0をうめこむ
02h	バイト境界まで1をうめこむ
03h	バイト境界まで端数を丸め込む
04~FFh	予約

ビット順列指定：画像データのホストへの転送順番です。この値の詳細はベンダ依存です。

圧縮引き数：画像データに作用する圧縮方法を与えます。

コード	圧縮方法
00h	圧縮しない
01h	CCITT グループ3 1次元(MH 圧縮)
02h	CCITT グループ3 2次元(MM 圧縮)
03h	CCITT グループ4 2次元(MMR 圧縮)
04~0Fh	予約
10h	OCR
11~7Fh	予約
80~FFh	ベンダ固有

■ Mode Select 命令の詳細

命令自体は SCSI の基本形なので省略します。ここでは Set Window に必要なユーザ単位の定義を説明します。ユーザ単位は、読み取り領域の全体のどの位置にどれくらいのサイズのウィンドウを設定するかを

図30 Object Position 命令

ビット バイト	7	6	5	4	3	2	1	0
0	OP コード (31h)							
1	LUN			予約		位置決め機能		
2	回数							
3								
4								
5	予約							
6								
7								
8								
9	コントロール							

決めるときに使用した単位でした。

スキャナではこの単位に、mm、インチ、ピクセルの3種から選択ができます。ただし、SCSI では浮動小数点を使用しないので、単位の定義は二つのパラメータを指定することになっています。この指定をする場所は、Mode Select のパラメータ・リスト (ページ・コード：03h) です。実際の値はユーザ単位を測定単位除数で除した値となります。たとえば基本測定単位に 01h (mm) をセットし、測定単位除数を 10 進で 100 と設定しておきます。ここで Set Window 命令の左上隅の X 座標に 10 進で 300 と指定すると $300/100=3$ となります。ウィンドウは読み取り領域の左から 3 mm の位置から始まることになります。

▷モード・セレクト測定単位定義のページ

バイト	説 明
0.7	PS・パラメータ・セーブ・ビット
0.6	予約
0.[5-3]	ページ・コード (03h)
1	パラメータ長 (06h 固定)
2	基本測定単位
3	予約
4~5	測定単位除数
6~7	予約

基本測定単位：以下に示すコードで単位を決める。デフォルトはインチ。

コード	単 位
00h	インチ
01h	mm
02h	ポイント
03~FFh	予約

測定単位除数：1 基本測定単位に等しいバイナリの値。デフォルトは 200。

図31 Scan 命令

ビット バイト	7	6	5	4	3	2	1	0
0	OP コード (1Bh)							
1	LUN			予約				
2	予約							
3								
4	転送長							
5	コントロール							

Object Position 命令の詳細

この命令は必須ではありません。しかし、自動連続読み取りを行うには必要です。命令を受けると、そこに指定された位置決め機能を同じく指定された回数だけ実行します(図30)。

Object Position 命令の項目説明

位置決め機能：

コード	動作
000b	現在セットされている原稿を排紙します。原稿がセットされていないときに本命令を受けてもエラーとはせず、ステータス Good を返します。ジャムなどで原稿が排紙できない場合にはチェック・コンディションでセンス・キーをメディア・エラーとします。
001b	原稿をベース・ラインまで取り込みます。本命令を受けたときに原稿がすでに取り込まれていてもエラーとはせず、ステータス Good を返します。もし取り込みをしようとしてできなかった場合にはチェック・コンディションでセンス・キーをメディア・エラーとしたうえで EOM ビットを立てます。
010b	原稿をベース・ラインから Y 軸方向へ動かします。移動距離は、命令にある回数フィールドをユーザ単位として計算します。もし指定距離の移動ができない場合や原稿が取り込まれていない場合にはチェック・コンディションでセンス・キーをメディア・エラーとしたうえで EOM ビットを立てます。
011b	原稿を現在の位置から Y 軸方向へ動かします。移動距離は、命令にある回数フィールドを2の補数とみなします。これが正ならばベース・ラインから遠ざかる方向へ、負ならばベース・ラインの方向に移動します。
100b	原稿を半時計回りに回転させます。回転角度は、命令にある回数フィールドの値を 1/1000 度とみなして求めます。

Scan 命令の詳細

Set Window, Mode Select 命令で動作モード、ウィンドウ定義の終了したスキャナに原稿をセットしたうえで Scan 命令を送ると実際のスキャンが行われます(図31)。

転送長：本命令に続いてホストからデータ・アウトされるウィンドウ・リストの長さです。ウィンドウ・リストは、ウィンドウ識別子(0 から 255 までの数値)の単純な並びです。

Read 命令の詳細

各種のデータをホストへ送り出します(図32)。

画像データを除くデータは、スキャンしたデータを加工するためにホストが読み出すものです。画像データは、スキャンで指定したウィンドウ・リストにのせられているウィンドウの読み取りデータが順次送られてきます。この順番などの具体的な規定はありません。

図32 Read 命令

ビット バイト	0	1	2	3	4	5	6	7
0	OP コード (28h)							
1	LUN			予約				
2	データ・タイプ・コード							
3								
4	データ・タイプ細目							
5								
6	転送ブロック長							
7								
8								
9	コントロール							

● Read 命令のデータ・タイプ・コード

コード	説明
00h	画像データ
01h	ベンダ固有
02h	ハーフトーン・マスク・データ
03h	ガンマ補正データ
04h~7Fh	予約
08h~FFh	ベンダ固有

事例としては IS410 を参考にしてください。

3

実機に見るスキャナ機能の特徴と拡張機能

リコーのスキャナ IS410 を参考にして実際のスキャナがどのような SCSI 機能の実装をしているかを見ることにします(写真2)。ここでは、そのインターフェース仕様書に述べられている機能の一部を紹介します。なにしろ原本は A4 版で 120 ページもあるものですから、とても全部を解説することなどできませんが、これから SCSI でスキャナを利用するソフトを開発する方にとって役に立つような解説を試みるつもりです。

基本仕様

最大読み取り領域：297 mm×432 mm(A3 版相当)

読み取り分解能：60, 75, 100~800 DPI

読み取り時間：2.1 秒(A4 原稿タテ, 200 DPI)

データ種類：2 値, 多値(4, 8ビット・グレー・スケール)
 処理機能：画質処理(2 値ライン・アート, ディザ・ハーフトーン, 誤差拡散・ハーフトーン)
 バイナリ空間フィルタ(黒孤立点除去, 凹凸補正, エッジ抽出, 黒画像の太線化)
 ガンマ補正フィルタ 4 種
 コード圧縮(MH, MR, MMR 法)
 ミラー出力, 白黒反転出力, MTF 出力
 シングル・ウィンドウ(ウィンドウ ID0 のみ)
 セクション機能(ウィンドウの内部を最大 6 セクションまで区分けして 2 値化処理の方法をセクションごとに選ぶことができる),
 拡張補助機構：ADF(これは自動原稿フィーダのこと, プリンタなどの用紙フィーダと同じで, 何十枚もの原稿をつぎつぎとスキャンする場合に便利な機構, ちなみに本フィーダを使用する自動スキャンを ADF モード, 手で原稿をスキャン面に置く方法をブック・モードと呼んで区別する)エンドーサ(文字や記号を原稿に印字する機能のこと, FAX などでは, 原稿の送信が確実に行われたことを示すために送信後“済”という朱のスタンプを押す機能がある, 同様に ADF を使って無人で原稿を読ませておいた場合などスキャンを終了した紙にマークを印字してやれば原稿が ADF によって確実にロードされたかどうかははっきりする),

■ 基本インターフェース

シングルエンド伝送でコネクタは, アンフェノール 50 P です。ターミネータは外付けとなっており内蔵していません。デバイス・タイプは, ターゲットであって論理装置 0 のみが実装されています。ディスクネクスト/リコネクトができますから, 時間のかかるメカ動作



写真2 リコーのスキナ IS410

の最中にバスを開放することができます。

転送ハンドシェイクは, 非同期のみです。また, コマンド・リンク・ビットは使用できません。

IS410 独自の拡張機能は, SCSI の基本フレームに組み込まれているベンダ定義域にきれいに収まっています。このため, もっとも基礎的な動作を SCSI の規定範囲内で行わせるソフトであれば, このスキナは動作するであろうと思われます。蛇足ですが, このような互換性の高いソフトの作り込みは, OEM メーカーにとって今後ますます重要になるでしょう。なぜなら, 互換性の高いシステムは理解がしやすいという点にテストが容易だからです。

● 有効 SCSI 命令

つぎに IS410 で有効な SCSI 命令を示します。

Test Unit Ready, Request Sense, Inquiry, Mode Select, Reserve Unit, Release Unit, Mode Sense, Scan, Receive Diagnostic Results, Send Diagnostic, Set Window, Get Window, Read, Send, Object Position, Get Data Buffer Status

さきほどは規格の必須命令だけでスキナからデータをとる例を示しました。ここでは比較の意味で IS410 を実際に使うつもりで, IS410 のプログラミング・ガイドにある命令シーケンスを紹介します。IS410 プログラミング・ガイドにはいくつもの例示があり, OEM ユーザにとって良い指針になっています。その中から複数枚の原稿を ADF を使って連続自動読み取りする場合の SCSI 命令シーケンスを紹介します。

(1) Reserve Unit 命令でスキナの排他利用を宣言します。これは複数のイニシエータがスキナを使う可能性がない場合には不用です。

(2) Inquiry 命令で VPD ページを読み出して ADF の装着を確認します(図33 参照)。

(3) Mode Select 命令でスキナの給紙モードを ADF と指定します。

(4) Set Window 命令でウィンドウ(ID=0 の一つのみ)を定義します。定義には規格の範囲に加えて IS410 の固有機能も含まれます。

ここでホストは, ウィンドウの定義から読み取りデータの総バイト数を計算しておきます。

(5) Read 命令を実行します。転送ブロック長はウィンドウ・データの総バイト数を与えます。IS410 は Read 命令を受信すると ADF を作動させて原稿を 1

枚取り込み、スキャンを実行します。

ここからつぎのデータ・イン・フェーズまではスキャナの内部処理になります。つまり、光学的スキャンを実行しながら Set Window で指定した処理を実施します。基本的な処理としては、領域を切り出す、フィルタ処理を施す、指定があればデータを圧縮するなどです。処理の終了とともにデータ・イン・フェーズで画像データがホストに転送されます。転送されたデータは、図34のようなデータ・フォーマットになっています。

つぎにホストが Read 命令を行うと IS410 は原稿を排出しながらつぎの原稿を取り込みます。あとは同様の操作をすべての原稿に対して繰り返します。

(6) Object Position 命令で最後の原稿を排紙します。

(7) Release Unit 命令でスキャナを解放します。

終わりに

このスキャナの項の執筆にあたっては、(株)リコーの横山、高桑両氏のご協力をいただきました。誌面を借りてお礼申し上げます。

参考文献

- 1) 「IS410 プログラミングガイド V2.0」、(株)リコー DEP 事業部
- 2) 「リコーイメージスキャナ IS410 SCSI インターフェイス仕様 V1.6」、同上
- 3) ANSI X3T9.2 WORKING DRAFT 375R REV10k 17-MAR-93, ANSI

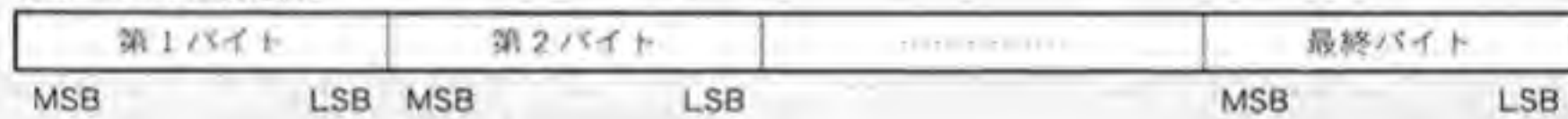
ふたがみ・たかお 前出

図34 リコー IS410 の非圧縮のデータ・フォーマット例(IS410 SCSI I/F 仕様書 V.1.6 より)

イメージ・データ・フォーマット

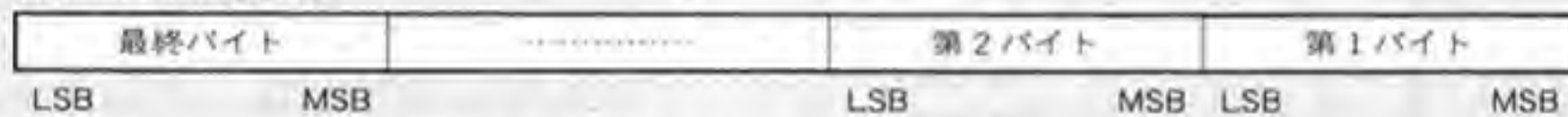
(1) 通常出力時

各ラインの出力順序



(2) ミラー出力時

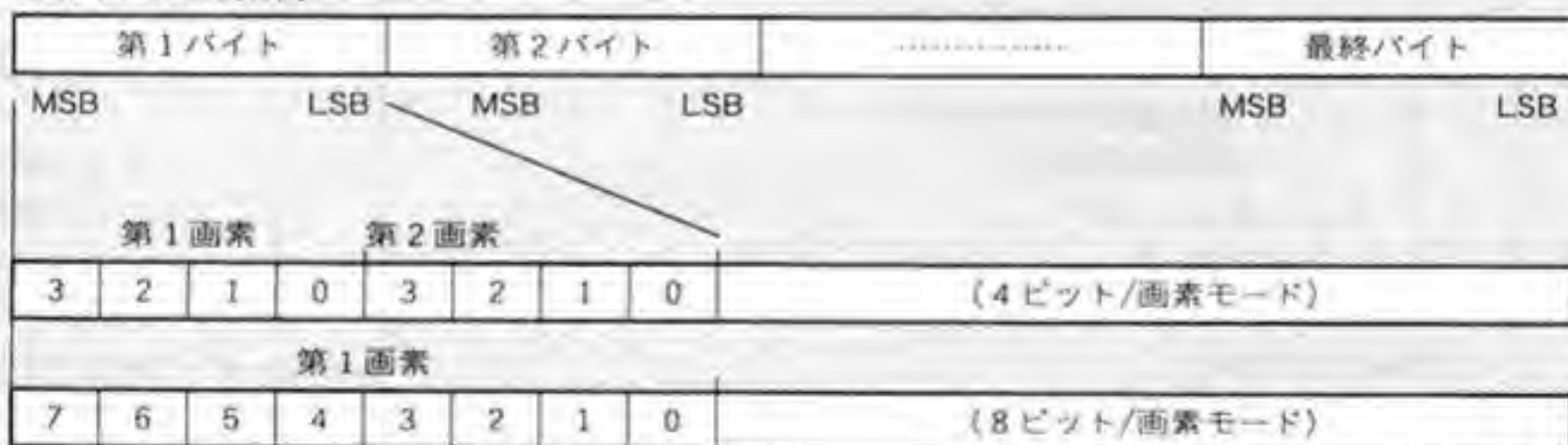
各ラインの出力順序



(a) 白黒2値(ライン・アート/ハーフトーン)

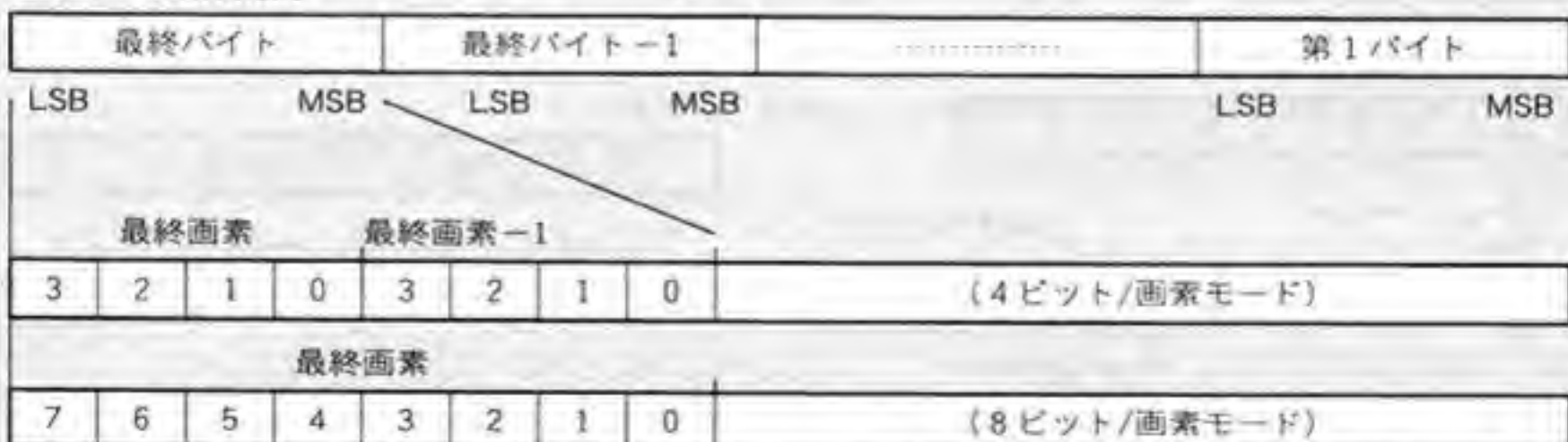
(1) 通常出力時

各ラインの出力順序



(2) ミラー出力時

各ラインの出力順序



(b) 多値

この VPD ページは、スキャナのオプション関係を表している。

この VPD ページは、スキャナのオプション関係を表している

--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--

© 2004 Blackwell Publishing Ltd, *Journal of Internal Medicine* 255: 105–112

© 2000 Blackwell Science Ltd, *Journal of Internal Medicine* 247: 399–404

コード 肉 容

Source: U.S. Census Bureau, *Marriage, Divorce, Remarriage in the 1990s* (Washington, D.C.: U.S. Government Printing Office, 1996).

コード 肉 窓

A10 SCSI I/E 仕様書 V1.6 上(1)

子切

100

--	--	--	--	--	--	--	--

3.4

QIC ストリーマ

大島 啓孝

QIC は“クイック”と読み、Quarter Inch Cartridge の略で 1/4 インチ・カートリッジ・テープのことですが、そのほかにも QIC 関連の企業で構成される世界的組織 Quarter Inch Cartridge Drive Standards Inc. を意味したり、この組織内の委員会で作られた記録フォーマットやインターフェースなどの規格の名称にもなっています。

QIC ストリーマは、世界市場で累計 900 万台以上の出荷実績をもちます。これは全テープ装置の 70 % 以上を占めていることになり、シーケンシャル・アクセス・デバイスの代表装置といえます。このような実績は QIC 委員会の活動に負うところが多いので、まず QIC の組織・活動についてすこしふれておきます。

1

QIC の組織と活動

QIC Drive Standards Inc. は QIC ストリーマ、媒体(テープ)、装置用部品(ヘッド、LSI 等)などのメーカーおよびユーザを含め約 40 社で構成される世界的な組織で(図35)、市場要求を十分に把握し、それをもとに性能・仕様の検討、規格統一化を行い、将来計画(マイグレーション・パス、図36)を作るとともに、現状製品の互換性をつねに監視し、また QIC の普及にもつとめています。

図35 QIC のメンバ

正会員 アイオーメガ・テープ
エクサバイト
ギガテック・メモリー・システム
コナー
コロラド・メモリー・システム
コムバイト
スリーエム
ソニー
タンベルグデータ
バーベイタム
マウンテンネットワーク・ソリューション
レクソン

技術会員 IBM
フィリップス
ヘラルド・データネティックス
准会員(抜粋) アダプテック
AHA
アイワ
データゼネラル
コダック
富士写真フイルム
日立マクセル
シマンテック
シーゲート
TDK
ティアック
ユニシス
ほか20社ほど
ほとんどが QIC 関連製品メーカーまたはユーザ

2

QIC ストリーマの動作概要

QIC ストリーマは容量面で、後発の DAT や 8 mm に遅れをとっていましたが、表 6 のように現在 5.25 インチ・フォーム・ファクタの装置で 2.5 G バイトのものが量産されており、1994 年の中頃から後半にかけて 5 G バイト、13 G バイトのストリーマが登場し、3.5 インチ・フォーム・ファクタ(ミニ QIC ストリーマ)でも 1 G バイトを超えるものが入手可能となります。このように、フォーマット(トラック数、記録密度など)は何種類もありますが、基本的には似たようなデータ構造をとっていますので、QIC-525 記録フォーマットを例にとって説明を進めます(QIC-525 より下位のフォーマットは ECC がなく、上位フォーマットはトラック数が多くなり図が見にくくなりますので)。

図36 5.25 インチ QIC ストリーマのマイグレーション・パス

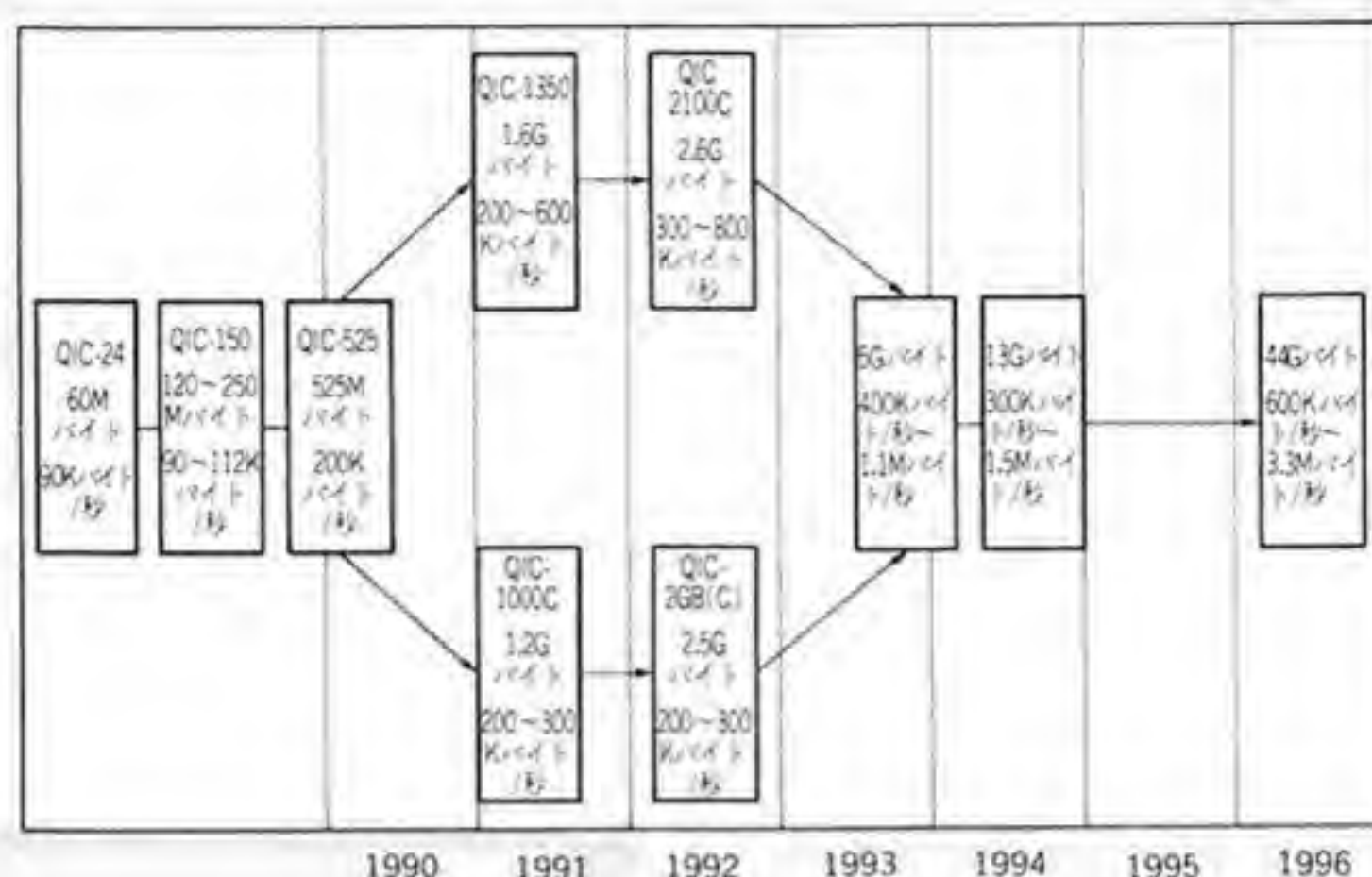


表6 QIC で規格化したおもなフォーマット(5.25 インチの場合)

記録フォーマット名	QIC-24	QIC-120	QIC-150	QIC-525	QIC-1000(C)	QIC-2GB(C)
テープ1巻あたりの容量(長尺テープ使用時)	60 MB	120 MB	150 MB (250 MB)	525 MB	1 GB (1.2 GB)	2 GB (2.5 GB)
トラック数	9	15	18	26	30	42
記録密度(1インチあたりの磁束反転)	10000 frpi	12500 frpi	12500 frpi	20000 frpi	45000 frpi	50800 frpi
データ密度(ビット/インチ)	8000	10000	10000	16000	36000	40640
平均データ転送速度(Kバイト/秒)	90	90	90	200	200~300	300
エラー修正機能	なし	なし	なし	リード・ソロモン、レベル2	リード・ソロモン、レベル2	リード・ソロモン、レベル2

■ メディアの構造

もともとデータ用に開発された QIC データ・カートリッジは図37のように動力部を除くほとんどのテープ走行系をカートリッジ内に構成しているので、テープ・パスも非常にシンプルで信頼性にたけています。また、じょうぶなアルミ・ベース・プレートとプラスチック・カバーによる準密閉構造なので、輸送や保管にも優れた性能をもっています。

■ メディア上のデータ、ブロック、トラック

テープ上のトラックは図38のようになっています。ヘッドは基本的に図39のようになっており、下側のチャネル(R_0 , W_0)が偶数トラック用、上側(R_1 , W_1)が奇数トラック用で、テープの終端でヘッドが上下に動き、

つぎのトラック位置につくようになっています。テープ・カートリッジを入れたときやリセット時、リワインド実行後などは、テープは BOT(ビギニング・オブ・テープ)側に位置し、書き込む場合、そこからトラック 0 に書き込み EOT(エンド・オブ・テープ)側に到着するとテープ走行を逆転し、トラック 1 に書き込みます。このように1本のテープを何トラックかに分け、行ったり来たりしながら記録する方法をサーペンタイン記録方式といいます。

QIC ストリーマは、現在までの規格ではオーバ・ライト(上書き)を許していません。イレース(消去)はテープ幅全体をいっぺんに行う方法で、Erase コマンドによっても消去できますが、通常の Write コマンドでもトラック 0 に書き込み中、自動的にイレースされる

図37 QIC データ・カートリッジとその駆動システム

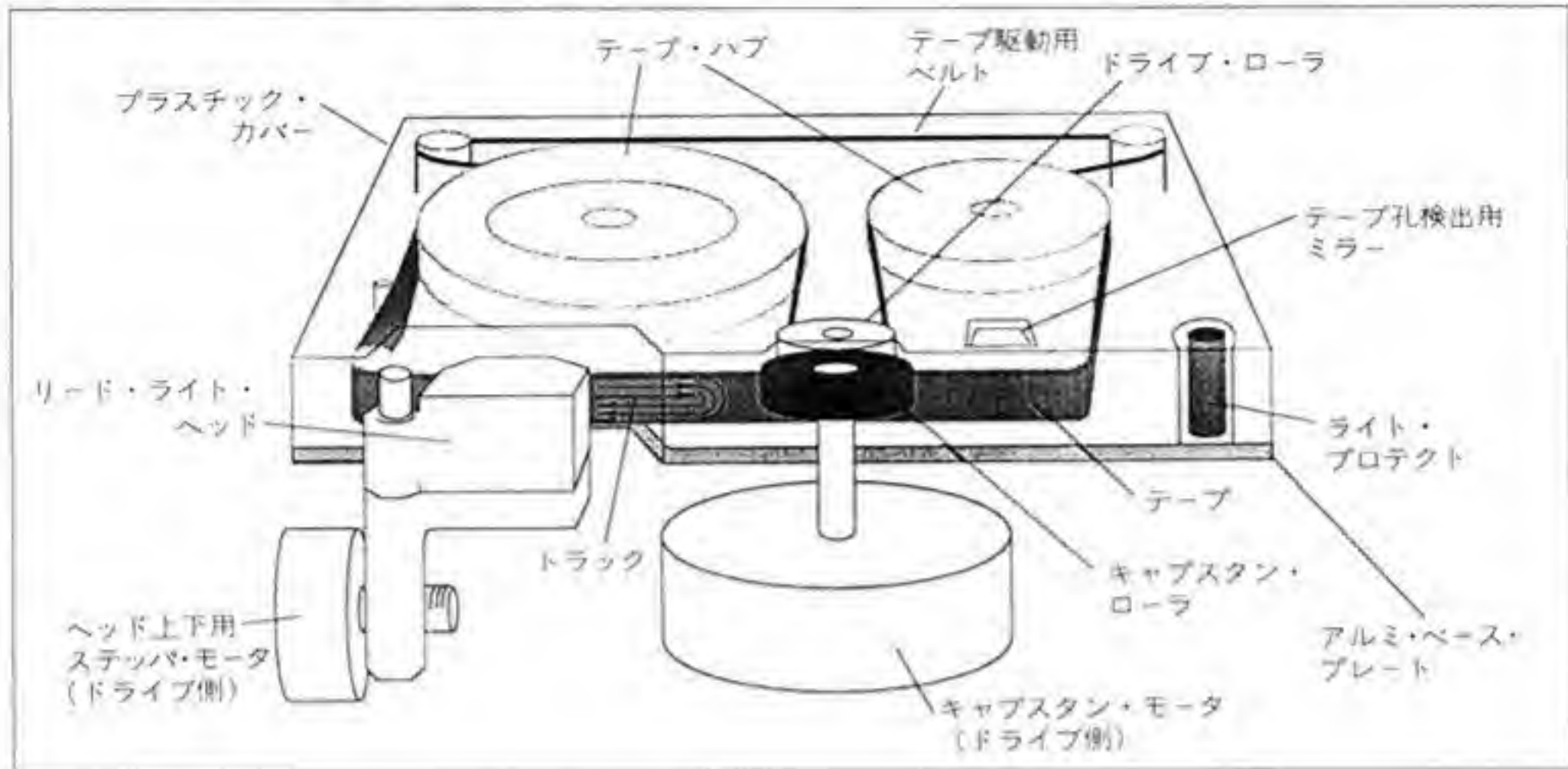
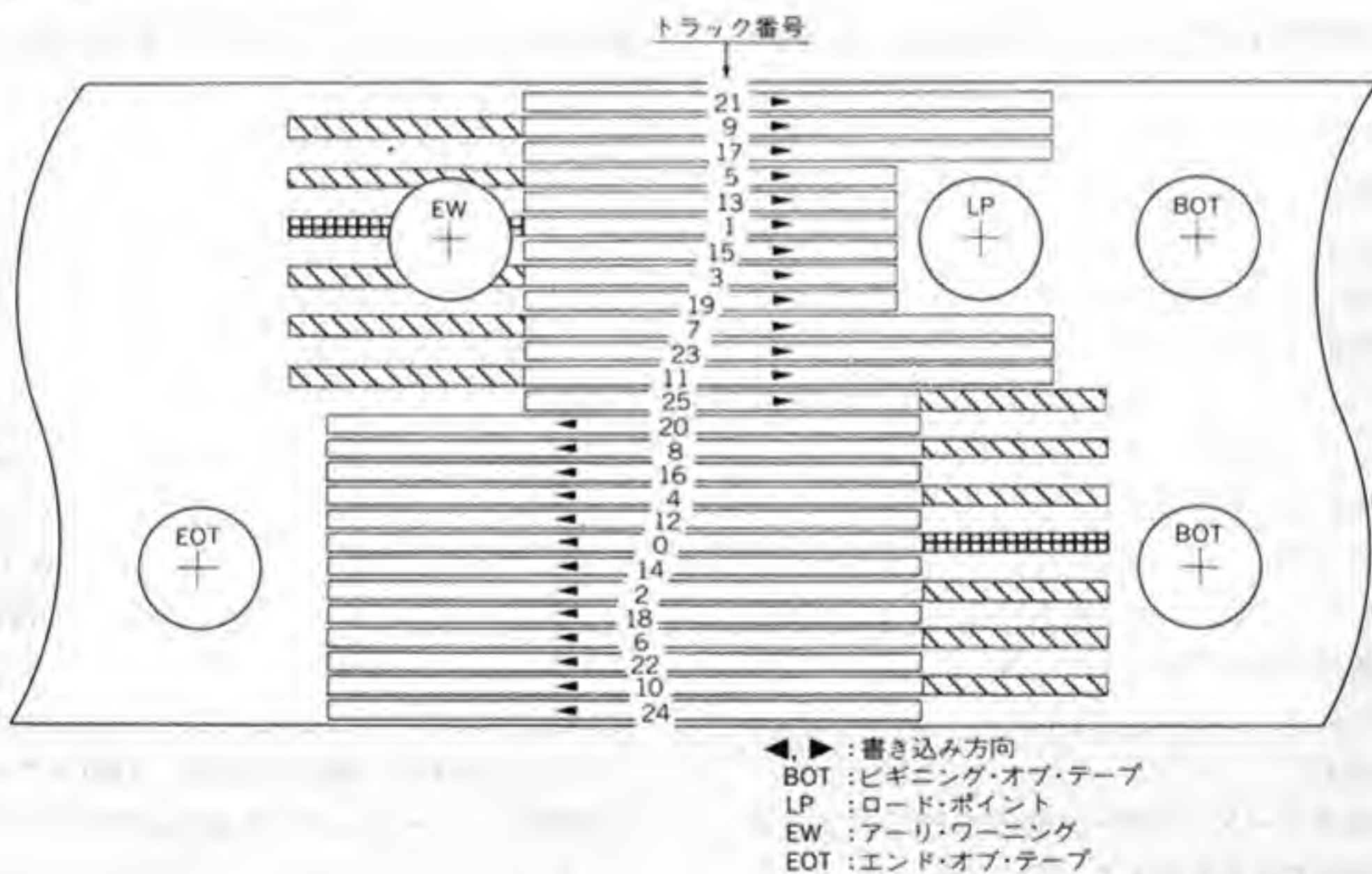


図38 QIC-525トラック・レイアウト



ので、マル秘データを消去する場合以外、とくに Erase コマンドを使う必要はないでしょう。

さて、トラックのなかを見てみましょう(図40)。データはブロックという単位で書き込まれ、管理されます。1ブロックの大きさはQIC-150までが512バイト、QIC-525以上は1024バイトです。ここでいうブロックはテープ上でのことで、SCSI上のブロックとは異なります。SCSI上では、1ブロック512バイトとか

1024バイト、またバリエーションでは1バイト単位でブロック・サイズが決められますが、テープ上では、フォーマットが決まればブロックの大きさは固定されます。これを知っているとメディアの有効利用に役立つと思います。つぎに、ブロックがいくつか集まって“フレーム”を構成します(QIC-525以上)。これはエラー修正機能(ECC)をもたせるためで、QIC-525の例では、データ・ブロック14個に2個のECCブロックが付加され

1 フレームとなります。データ・ブロックには、もちろん、ユーザ・データが入りますが、ECC ブロックは装置が自動的に作るため、ユーザには見えない「オーバヘッド」となります。この ECC の訂正能力は大変強力で、1 フレーム内に完全に壊れたブロックが2 個あってもデータの修復が可能で、QIC-525 の場合、約 27 mm 長のスクラッチができて大丈夫ということです。

ECC ブロックのようにユーザに見えないブロックは何種類かあります。たとえば、書き込みの終了がフレーム境界にない場合に余りをうめるために装置が発生するフィラー・ブロックや、装置自身が使うための制御用ブロックがあります。ユーザに見えるブロックはデータ・ブロックとファイルマーク・ブロック(およびセットマーク・ブロック)です。ファイルマーク・ブロックはユーザがファイル管理に使うためのもので、ファイルの“区切り”として使用できます。ファイルマークの書き込みは Write Filemarks コマンドで行いますが、CDB 内でセットマークを指定(オプション)することもできます。これにより、ファイルマークとセットマークという2 種類のマーカをテープ上に書き標することができます。リード時は、ファイルマークまたはセットマークを通過すると装置は Check Condition を発生します。また Space コマンドにより、ファイルマークやセットマークをさがすこともできます。

■ エラー・リカバリ

磁気媒体にはかならず欠陥(ディフェクト)があります(メディア屋さんゴメンなさい!!)。したがって、装置がこれをうまく処理する手段をもっていなければ、データの信頼性が損なわれます。QIC ストリーマではどのようにエラー・リカバリを行っているのかを簡単に説明します。

リード時のリード・エラーは前述の ECC によるエラー修正機能で処理されます。ECC を使ってもダメな場合はリード・リトライが行われエラーに対処します

が、なんとか読もうとリトライの際にリード・ゲインを変えたり、トラック位置をずらしたりしますが、方法はメーカーにより若干異なるようです。ライト時は、図39 のヘッド構造からもわかるように、テープはライト・ギャップを通過時に書き込まれ、その後リード・ギャップを通過する際、リードされ正しく書き込まれたかどうかチェックされます。もし、あるブロックが正しく書き込まれていなかった場合、同じブロックをもう一度書き込む、ライト・リトライを行います。ライト・ギャップとリード・ギャップの距離が1 ブロックの長さより大きいため、あるブロックのエラーを発見したときはすでにつぎのブロックを書いているので、実際のライト・リトライは図41 のように行われます。

■ 特徴的なコマンド

今までの説明にでてきた Erase, Write Filemarks, Space はすべてシーケンシャル・アクセス・デバイス独特のコマンドです。もうすこし Write Filemarks と Space について説明しましょう。

▶ Write Filemarks

ファイルマークやセットマークを書き込むコマンドですが、「連続していくつ書くか」も指定します。これらのマーカの数と Space コマンドの組み合わせによって、細かいファイル管理も可能です。

▶ Space

通常の使い方は、テープの現在位置から n 個目のブロックかファイルマーク(またはセットマーク)へ移動させる(テープを走らせる)コマンドです。探す対象(ブロック、ファイルマーク、セットマーク)の指定と何番目か(n)の指定が必要で、 n は負の数でもかまいません。負の数はリバース方向に探すことを意味します。

ちょっと変わった使い方では、 n 個(またはそれ以上)連続したファイルマークまたはセットマークを探すこともできます。ただし、この場合は“(n 個の)何番目の”という指定はできません。

図39 ヘッド正面

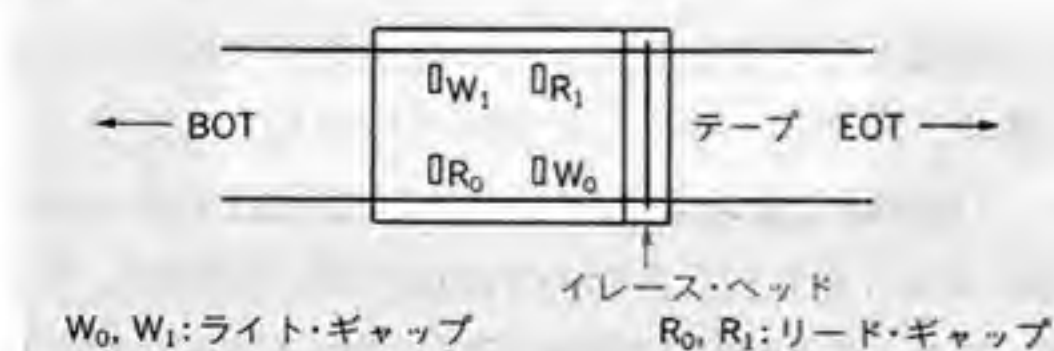
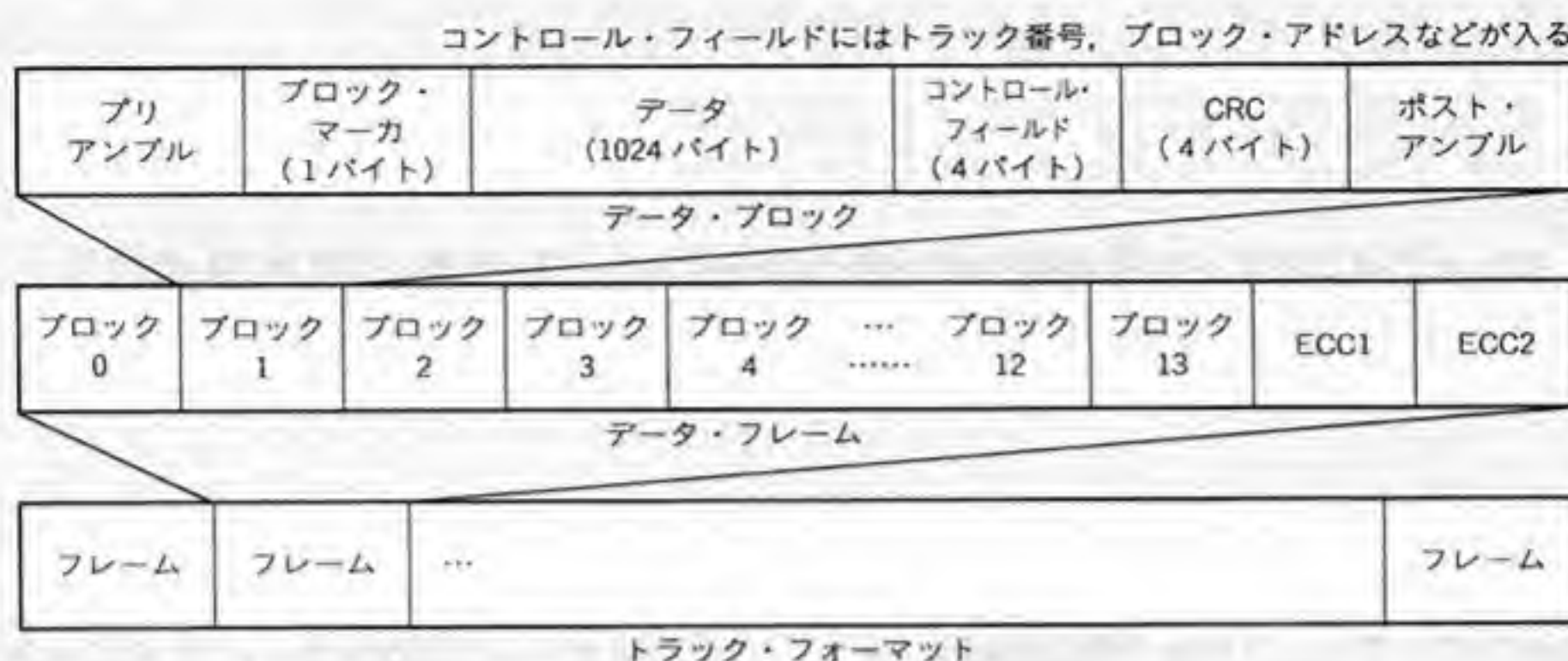


図41 ライト・リトライ方法



図40 ブロック、フレームの構造



■ QFA(Quick File Access : クイック・ファイル・アクセス)

QIC ストリーマの特徴的な使い方で QFA というものがあります。QIC ストリーマは、ビデオやオーディオ用ではないので、8 mm や DAT のような何倍速サーチのような機能は元来もっていません。しかし、もともとがデータ記録用である特徴を生かし、ファイル・サーチを高速にするのが、この QFA です。この考え方は非常に論理的で、装置自身も持っている「特定ブロックへの移動アルゴリズム」を使う Locate コマンドとパーティションの考え方を使うものです。QIC の記録方式はサーペンタインなので、“トラック”という大きい物理的な単位でパーティションを切ることが

できます。QIC では各フォーマットでの最終のトラックを別のパーティションとし、そこをディレクトリ領域とすることにより、フロッピーやハード・ディスクのようなファイル管理を行い、希望のデータを素早くアクセスする方法を“QFA”として推奨しています。QFA を使う場合、一つのトラックを別パーティションとするわけですが、「パーティションの始まりは BOT から」と決められています。そこで、たとえば QIC-525 で通常記録では最終トラック(トラック 25)は EOT→BOT 方向の記録ですが、QFA を使用した場合、トラック 25(実際にはデータとは別のパーティションのトラック 0)の書き込み方向は BOT→EOT となります。

おしま・ひろたか 前出

機種別 SCSI 研究

98・AT・UNIX/他機種用がなぜつながらない/互換性の阻害要因は？/
デバイス・ドライバはどうつくる

本来、SCSI インターフェースをもつ周辺機器は、ホスト・コンピュータの機種がなんであろうとつながるはずだが、現実には「××用ハード・ディスク」といったように、ホスト・コンピュータが指定されている。だから、パソコン用の大容量ディスク装置が遊んでいるからといって、これをワークステーションに流用しようとしても、簡単にはいかない。同じパソコン用のディスク装置でも、たとえば PC-9801 用を IBM AT に接続するのはかなり難しいのが実状である。「なんにでもつながるのが標準インターフェースじゃないか」という声が聞こえてきそうだが、コネクタ仕様の不一致などの物理的原因からはじまって、SCSI コマンドのサポートの違いや、プロトコルの問題など、さまざまな要因があって、実際には、機種依存が厳として存在する。そこで、▶まず最初に、互換性の問題でなにかと話題にのぼることが多かった PC-9801 の

SCSI ボード(55&92 ボード)に焦点をあて、互換性の阻害要因を、市販の SCSI モニタや自作のユーティリティを駆使しながら解析する。▶ついで、互換性の面では比較的うまくいっている IBM AT および互換機の SCSI をとりあげる。AT 互換機の世界では、各種の SCSI アダプタ・ボードごとにデバイス・ドライバを開発せずにすむように、アダプテック社が ASPI(Advanced SCSI Programming Interface)という仕様をきめたが、現在これが事実上の標準になっている。そこで、ここでは、まず、SCSI と ASPI の関係を調べ、ついで、ASPI によるプログラミングを紹介する。▶最後が、UNIX である。UNIX の例として、SunOS に着目し、SCSA(Sun Common SCSI Architecture)を紹介しながら、難しいといわれる UNIX の SCSI ドライバ作りに挑戦してみる。(編集部)

4.1

98のSCSIと55&92ボード

真樹 美智

PC-9801 シリーズ用のハード・ディスクがハード・ディスク専用インターフェースである SASI から、MO、CD-ROM なども接続できる SCSI インターフェースへと変遷し、現在では DOS/V を意識してか低コスト・高速アクセスの IDE インターフェースを搭載した 98MATE、98FELLOW も登場してきました。

IDE は、はじめ IBM パソコンの内蔵ハード・ディスクに使用されていたもので、NEC でも 98 ノートの 2.5 インチ内蔵ハード・ディスクのインターフェースとして採用されているので新しい話題ではありません。IDE はケーブル長をあまり延ばせないことやプロト

コルの単純さから、おもに内蔵ハード・ディスクに用いられています。一方、拡張性に優れている SCSI は今後とも外付け周辺機器のインターフェースの主役でありつづけそうです。

しかし、拡張性に優れている SCSI ですが、PC-9801 の SCSI では、互換性の問題がありました。PC-9801-55 ボード(通称 55 ボード)が Inquiry データの“NEC”にプロテクトをかけ、純正ハード・ディスク以外のハード・ディスクを接続できないようにしていたことは、よく知られています。

Inquiry データの“NEC”にプロテクトをかけた 55

図1 PC-9801用SCSIボードの構成

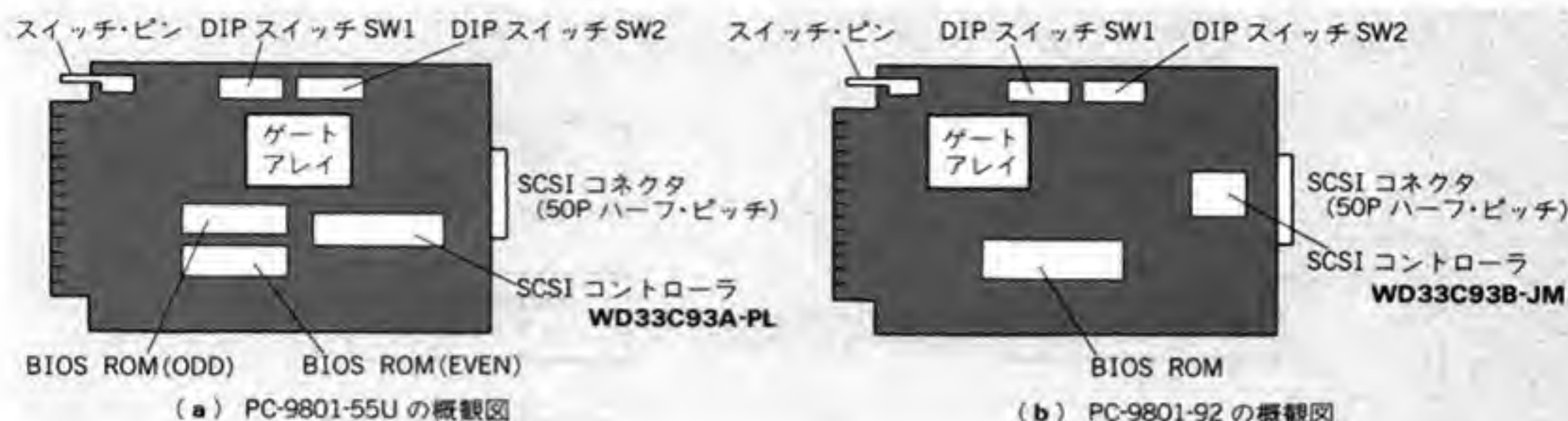
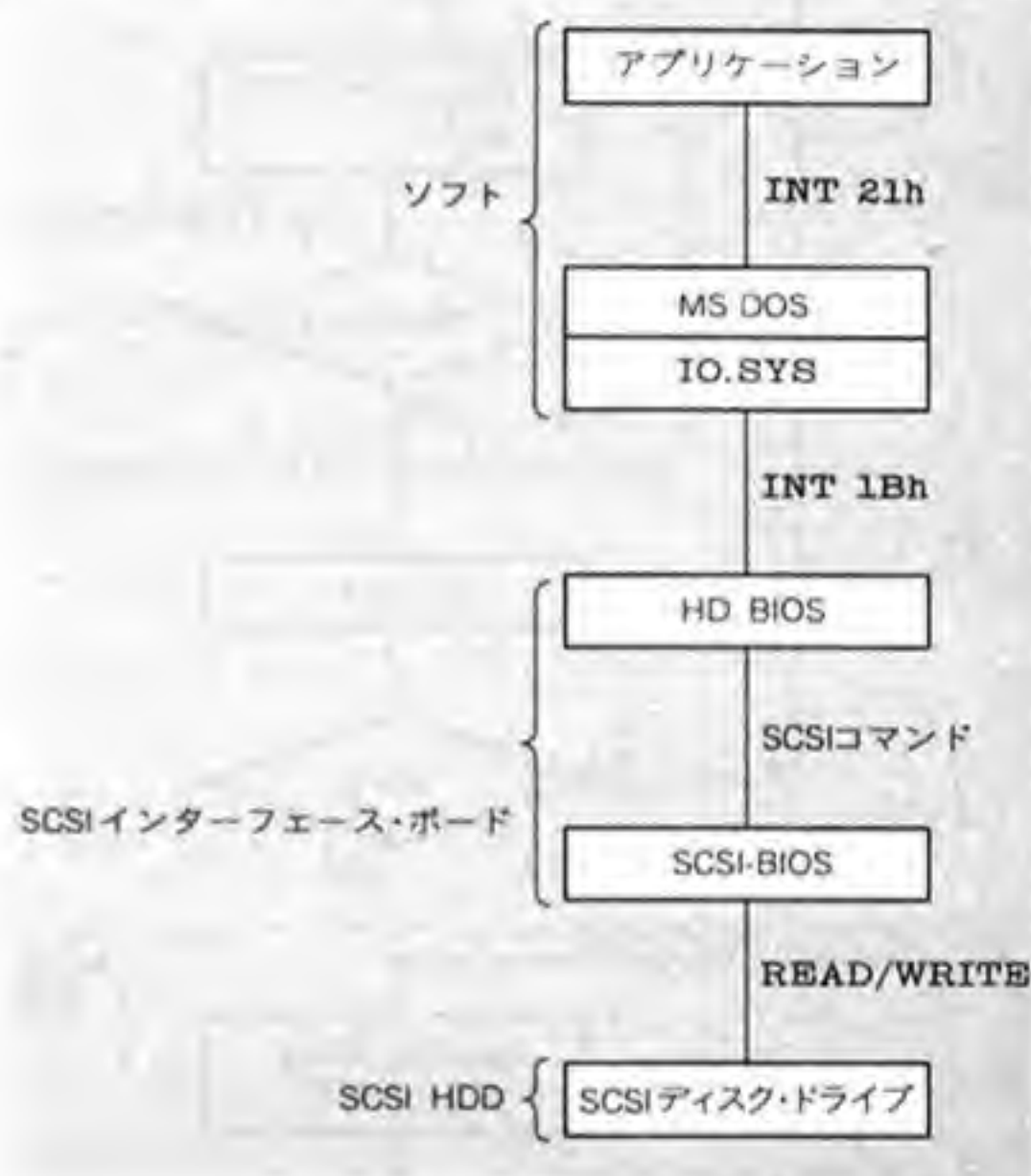


図2 PC-9801+MS-DOSの制御ブロック図



ボードについて'93年7月に発表されたPC-9801-92ボードになって、やっとプロテクトがとれ、SCSI本来の姿になった観があります。ここでは55ボードが、なぜプロテクトをかけたのか、また92ボードではどう変わったのかを解析していきます(図1)。解析の手がかりを得るためには、市販のSCSIモニタとつぎの5章で紹介するSCSIユーティリティSU.EXEを活用しています。

1

なぜ55ボードでは他社製SCSIハード・ディスクが認識されないか

■ SCSIハード・ディスク立ち上がりシーケンス

PC-9801で、SCSIハード・ディスクを使う場合、どのサード・パーティのインターフェース・ボードでも、NEC純正のPC-9801-55ボードと、少なくともBIOSレベルで互換がとれています。MS-DOSを例にとるとアプリケーション側でファイルをアクセスする際に発行するファンクション・コール(INT 21h)はインターフェース・ボードを経由して最終的にSCSIコマンドを発行します(図2)。

認識されない原因を説明する前に、PC-9801でのSCSIハード・ディスクの立ち上がりシーケンスがどのようなになるかを順を追って説明します(図3)。この実験をするために、NECのSCSIインターフェース・ボード(NEC PC-9801-55U)、45 Mバイト純正ドライブ(D3835)、およびSCSIモニタを使いました。

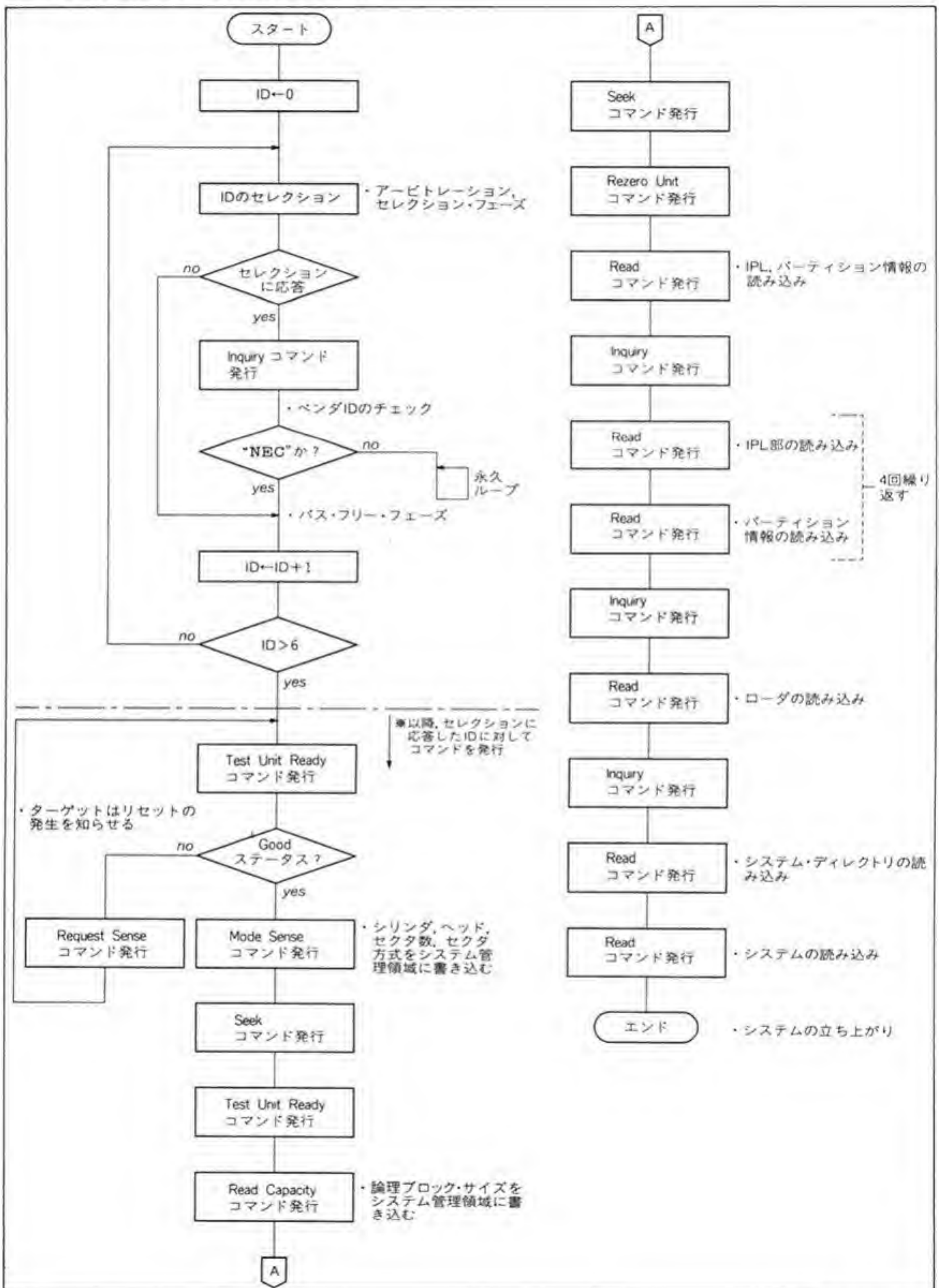
(1) イニシエータは、SCSI ID 0から6の順でセレクションを行い、応答があった装置に対してInquiryコマンドを発行する。応答がなければバス・フリーとし、

アービトレーション、セレクションを繰り返す。Inquiryデータの8バイト目からのベンダIDが“NEC”となっていないければ、ここから先に進まない。また、Inquiryデータから装置の機種(ハード・ディスク、光磁気ディスクなど)の判別を行う。

(2) イニシエータは、応答があった装置にTest Unit Readyコマンドを発行する。このとき装置はパワー・オン・リセット、またはバス・デバイス・リセットが発生したことを知らせるためにCheck Conditionステータス(02h)で応答する。

(3) イニシエータは、Check Conditionステータスに対しRequest Senseコマンドを発行し、センス・キー

図3 システム立ち上がりのSCSIコマンド・フロー



と追加センス・コードを得る。このとき、センス・キーの内容(06h)は“Unit Attention”で、追加センス・コードの内容(ASC:29h, ASCQ:00h)は, “Power on Reset, or Bus Device Reset Occurred”である。

(4) イニシエータは、再度 Test Unit Ready コマンドを発行する。装置は Good ステータス(00h)で応答する。

(5) イニシエータは、装置がハード・ディスクであれば Mode Sense コマンドを発行し、すべてのページ(ページ・コード 3Fh)を得る。D3835 ドライブが返送したページは、

ページ 1: Read-Write Error Recovery Page

ページ 3: Format Device Page

ページ 4: Rigid Disk Drive Geometry Page

ページ 8: Caching Page

ページ 3 からトラック当たりのセクタ数、セクタ方式、ページ 4 からシリンダ数およびヘッド数を 0:460 h から始まる SCSI 管理領域(仮称:図 4)に書き込む。このとき書き込まれたシリンダ数は、ページ 4 から読んだシリンダ数-1(IPL, パーティション情報域の 1 シリンダ分を除く)である。

また、0:482h のバイト(システム共通域情報:図

5)に該当する SCSI ID のビットをたてる。

(6) イニシエータは、論理ブロック・アドレス FFFFh に Seek コマンドを発行する。

(7) イニシエータは、Test Unit Ready コマンドを発行する。装置は Good ステータス(00h)で応答する。

(8) イニシエータは、Read Capacity コマンドを発行し論理ブロック・サイズを得て、前述の SCSI 管理領域に書き込む。このとき書き込まれた値は、32 ビット長の論理ブロック・サイズを右に 9 ビット・シフトした後の LSB の 2 ビットと等価である。

<例>

00000100h(論理ブロック・サイズ 256 バイト) → 00b

00000200h(論理ブロック・サイズ 512 バイト) → 01b

(9) イニシエータは、論理ブロック・アドレス 0000h に Seek コマンドを発行する。

(10) イニシエータは、Rezero Unit コマンドを発行する。

(11) イニシエータは、Read コマンドで論理ブロック・アドレス 0 から 2 ブロック分(IPL 部、パーティション情報部:図 6)を読む。パーティション情報からブート可能かどうかを判断し、IPL の起動/非起動を決定する(図 7)。



SCSI ハード・ディスク製品の現状

コンピュータ関連の雑誌などの広告を見ると、PC-9801/EPSON 用、Macintosh 用、FMR 用 SCSI ハード・ディスクというように、各コンピュータ専用の SCSI ハード・ディスクが、さも存在するように書かれています。

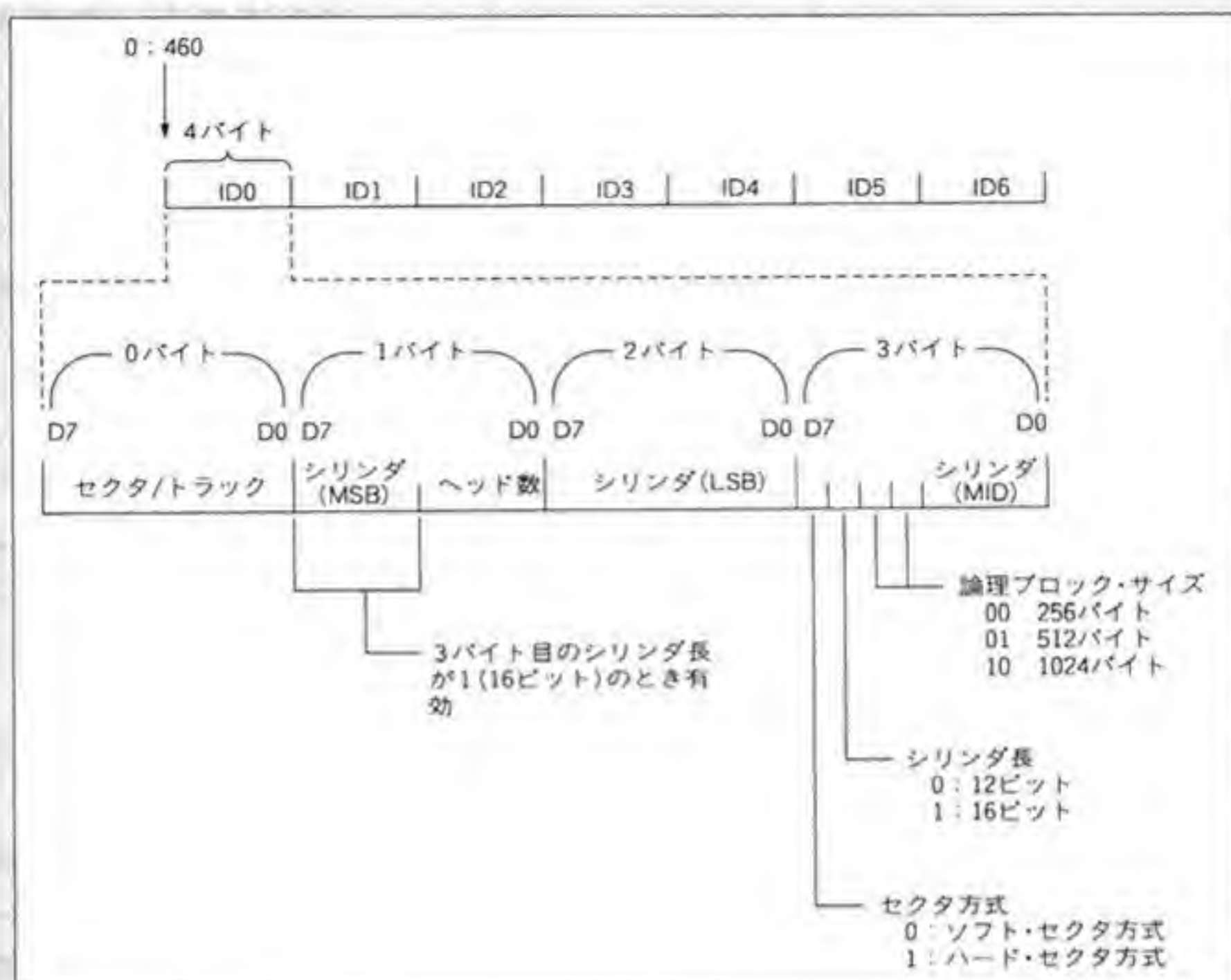
ところが ANSI の SCSI 規格書を調べても、「××メーカー用 SCSI 規格」という項目はどこにもなく、「××用 SCSI ハード・ディスク」とは、よほど厳密な機能を規定しているコンピュータ用でもないかぎりハード・ディスクそのものではなく、製品としてのことだと思ひ当たります。

つまり、PC-9801 のようにイニシエータにあたるコンピュータにもともと SCSI インターフェースを実装していなければ、ハード・ディスク・メーカーがこれを用意したうえで、はじめて製品として出荷できますし、Macintosh のように SCSI インターフェースは付属し

ていても、Mac 付属のインストーラ(ホスト・コンピュータにハード・ディスクを認識させるソフト)は純正ハード・ディスクとして登録していないものは無視してしまうので、独自にインストーラを用意して製品としています。FMR、Panacom M のようにハード・ディスクの物理フォーマット・ユーティリティが用意されていなければ、製品化以前にホストに最適なインターリーブ値で物理フォーマットを施す必要があります。また、ハード・ディスクの設計をホストと違和感ないようにすることも製品化の一つでしょう。

忘れてならないのは、メーカーのサポート体制で、仮に前述のすべてをクリアしたとして、コンピュータと OS と SCSI ハード・ディスクのあらゆる組み合わせで検証し、サポートすることは現状では無理があるかもしれません。

図4 SCSI 管理領域(諸元領域)の構造



- (12) イニシエータは、なぜか Inquiry コマンドを発行する。
- (13) イニシエータは、なぜか(11)で発行したはずの論理ブロック 0, 1 に対して Read コマンドを 4 回連続して発行する。
- (14) イニシエータは、なぜか Inquiry コマンドを再発行する。
- (15) イニシエータは Read コマンドを発行し、二つ目のシリンダの先頭から 2 ブロック分のシステム・ローダを読み込む。
- (16) イニシエータは、Read コマンドを発行し、IO.SYS, MSDOS.SYS, COMMAND.COM のディレクトリ部を 1 ブロック読み込む。

- (17) イニシエータは、Read コマンドを発行し、IO.SYS, MSDOS.SYS, COMMAND.COM ファイルを読み込み、システムが起動する。

■ 認識されない原因

立ち上がりシーケンスの(1)ですでに書いてしまいましたが、Inquiry コマンドを発行して、返送データの 8 バイト目からのベンダ ID が“NEC”となっていなければ、ここから先に進まないのが直接の原因です。重要な問題は、なぜ他社製をここではねるかということです。図 6 のパーティション情報をみればわかりますが、論理ドライブを区切る単位をシリンダとしていることに注目してください。シリンダを単位とした場合、シリンダ当たりの論理ブロック数を得るため、ヘッド数、トラック当たりのセクタ数もわかっていなければなりません。これらの構成を知るため、(5)で Mode Sense コマンドを発行してページ 3 (Format Device Page)、ページ 4 (Rigid Disk Drive Geometry Page) を得ています。図 8 は今回のユーティリティで D3835 の Mode Sense と Read Capacity コマンドを発行して

図5 システム共通情報

0:482 SCSIハード・ディスクが接続されていれば

ID6	ID5	ID4	ID3	ID2	ID1	ID0
ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	ビット0

図6 パーティション情報

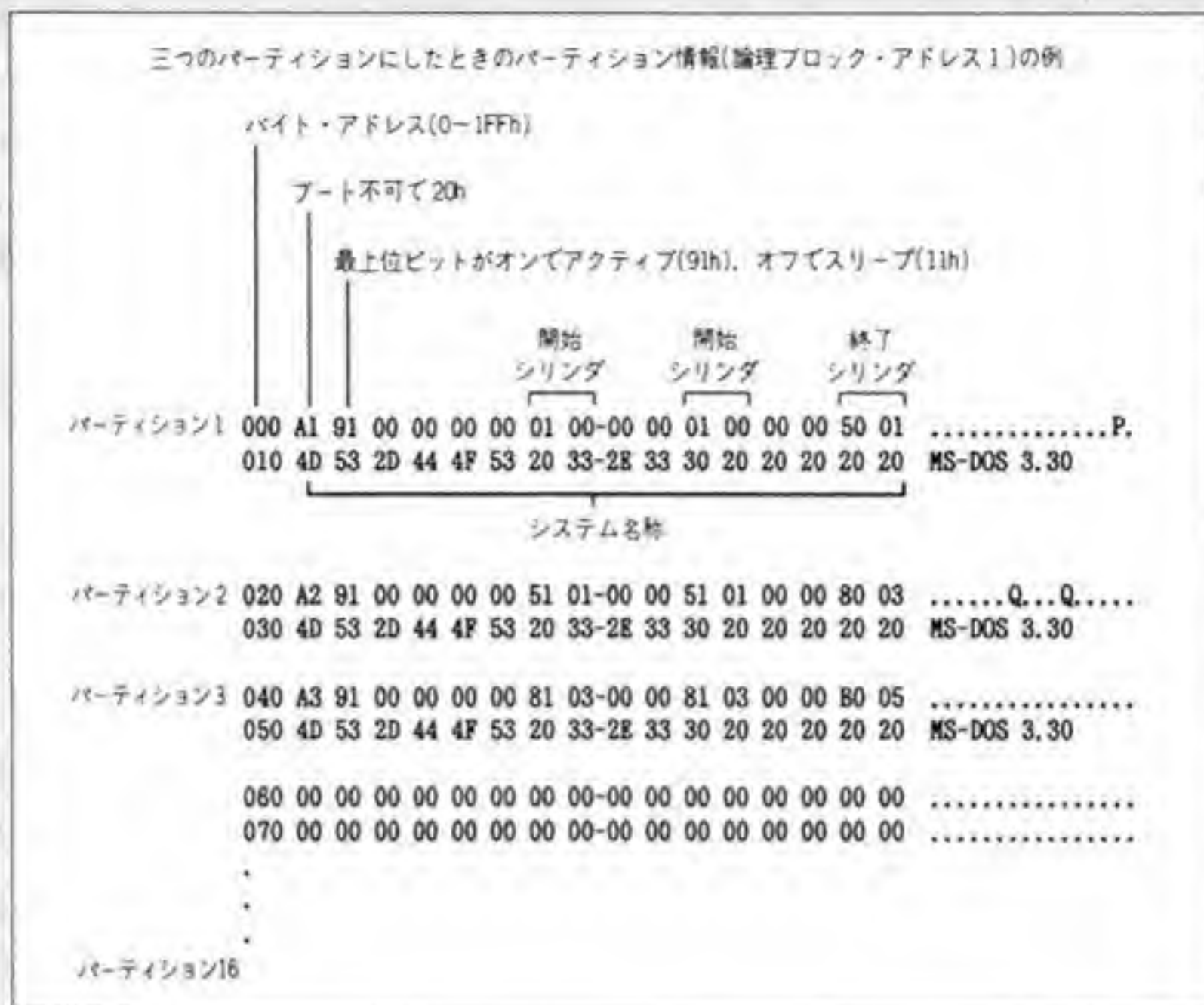


図7 ハード・ディスクの概念図

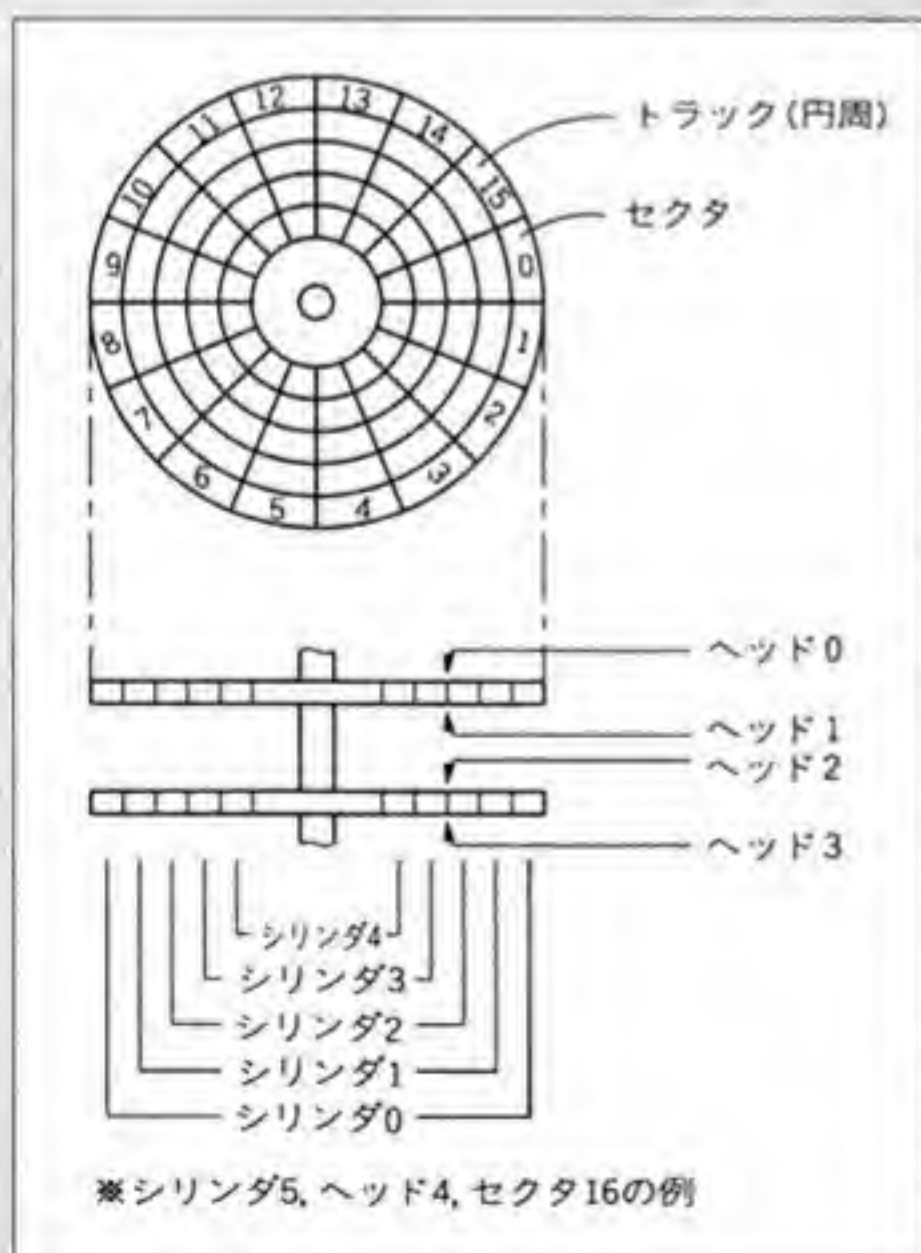


図8 □3835の緒元データ(5章 出力リストの説明を参照)

緒元データ	
バイト容量	: 45056 バイト
論理ブロック数	: 88000①
論理ブロックサイズ	: 512 バイト
物理セクタサイズ	: 0 バイト
シリンダ数	: 440②
ヘッド数	: 8③
セクタ/トラック	: 25④
トラック/ゾーン	: 8
交代セクタ/ゾーン	: 8
交代トラック/ゾーン	: 0
交代トラック/論理ユニット	: 80
インタープ・フォーマット	: 1
トラック・スキュー・フォーマット	: 0
シリンダ・スキュー・フォーマット	: 0
ソフトセクタ方式	: YES
ハードセクタ方式	: NO
リム・バブル	: NO
フォーマット	: シリンダ順
論理構成: シリンダ=440, ヘッド=8, セクタ/トラック=25, ブロック=88000	

合成した緒元データ・リストです。①の論理ブロック数を得るために、②のシリンダ数、③のヘッド数、④のセクタ/トラックを乗算してみましょう。

$$440(\text{シリンダ数}) \times 80(\text{ヘッド数}) \times 25(\text{セクタ/トラック}) = 88000(\text{論理ブロック数})$$

うまいこと論理ブロック数と一致したのがわかります。したがって、Mode Sense コマンドで得たデータをそのまま図4のSCSI管理領域に書き込むことが可能です。SYMDEBでSCSI管理領域をみると、Mode Sense データで得た値がそのまま書かれているのがわかります(図9)。

では、他社製のハード・ディスクではどうでしょうか。図10は、アルプス電気のDRR100Dの緒元データ・リストです。D3835と同様な計算をしてみると、

$$1468(\text{シリンダ数}) \times 4(\text{ヘッド数}) \times 72(\text{セクタ/トラック}) = 422784(\text{論理ブロック数})$$

論理ブロック数が205100ですから倍以上ふえてしまいました。DRR100Dの場合、⑤の物理ブロック・サイズが256バイトと、論理ブロック・サイズの2分の1のサイズになっていますので、④のセクタ/トラック

を論理セクタ・サイズに合わせるには2分の1にしなければなりません。また、交代処理の単位である⑥のトラック/ゾーンの4トラック中、⑦の交代セクタ/ゾーンが8であることから1トラック中に二つの交代物理セクタ(1論理セクタ)が含まれているので、トラック当たりの有効セクタ数を算出するには、この分を引く必要があります。DRR100Dの論理的なセクタ/トラックを便宜上算出してみると、

$$(72-2) \times 0.5 = 35(\text{論理セクタ/トラック})$$

また⑧の交代トラック/論理ユニットが12でヘッド数が4であることから、3シリンダ分を②のシリンダ数から引いて、有効シリンダ数を求めます。

$$1468-3 = 1465(\text{有効シリンダ数})$$

こうして物理緒元から論理構成を算出しておすと⑨の論理構成となり、論理ブロック数が一致します。

$$1465(\text{有効シリンダ}) \times 4(\text{ヘッド}) \times 35(\text{セクタ/トラック}) = 205100(\text{論理ブロック数})$$

こうした物理緒元から論理構成を算出しておすと⑨の論理構成となり、論理ブロック数が一致します。

図9 SYMDEBでSCSI管理領域をみる

```
A>SYMDEB
Microsoft Symbolic Debug Utility
Version 3.01
(C)Copyright Microsoft Corp 1984, 1985
Processor is [80286]
~db 0:480,483
0000:0460 19 08 B7 11
~q
A>

SCSI管理領域の値

シリンダ数 = 1B7h(439) .....1シリンダ少ないのはIPL、
ヘッド数 = 8 .....パーティションの分
セクタ/トラック = 19h(25)
```

図10 DRR100Dの緒元データ(5章 出力リストの説明を参照)

緒元データ		
バイト容量	: 105011 キロバイト	
論理ブロック数	: 205100①
論理ブロックサイズ	: 512 バイト	
物理セクタサイズ	: 256 バイト⑤
シリンダ数	: 1468②
ヘッド数	: 4③
セクタ/トラック	: 72④
トラック/ゾーン	: 4⑥
交代セクタ/ゾーン	: 8⑦
交代トラック/ゾーン	: 0	
交代トラック/論理ユニット	: 12⑧
インクリメント・ファクタ	: 1	
トラック・スキップ・ファクタ	: 0	
シリンダ・スキップ・ファクタ	: 0	
ソフトセクタ方式	: YES	
ハードセクタ方式	: NO	
リム・パブル	: NO	
シー・フェイス	: シリンダ順	
論理構成:シリンダ	=1465,ヘッド=4,セクタ/トラック=35,ブロック=205100	...⑨

DDR100D の場合、⑥のトラック/ゾーンの交代替理の単位が4トラック(1シリンダ分)であること、⑦の交代セクタ/ゾーン8セクタがヘッド数4で割り切れる(図11)ことから、きれいに論理構成を算出することができましたが、すべての SCSI ハード・ディスクがこのような値を返してくるとはかぎりません。SCSI の規格には、トラック/ゾーンの単位をシリンダで割り切れるとか、交代セクタ/ゾーンがヘッドで割り切れなければならない、などの事項はないからです。

ハード・ディスクによってはシリンダ中の有効セクタがヘッドで割り切れないものもあります(図12)。NEC の D3835 ではシリンダ、ヘッド、セクタ/トラックの乗算で論理ブロック・サイズが求められようになっていますが、SCSI-2 規格にはセクタ/トラックには交代セクタが含まれていることが書かれています。Mode Sense データをそのまま使っていれば、他社製ハード・ディスクと互換がとれなくなって当然です。このことが Inquiry コマンドで他社製ハード・ディスクをはねる原因のひとつと推察します。

たしかに SCSI ハード・ディスクが出始めのころは、Mode Sense データの書式も各社まちまちでした。というのも、従来のコンピュータがハード・ディスクをコントロールする場合、シリンダ、ヘッド、セクタ/トラックの値を直接操作することがホストの役目でした

が、SCSI ではハード・ディスク自身がこの役割をし、ホストはたんに論理ブロック番号で読み書きできるようになりました。したがって、ホストのファイル・システムも論理ブロック単位でアクセスすればよく、あえて緒元を知る必要がなくなってきたためです。

緒元データを必要としないホストは、Read Capacity のデータからブロック数と論理ブロック・サイズを得るだけで事足りるはずですが、このことは SCSI を採用しているいくつかのホストが論理ブロック番号でパーティション(コラム参照)を区切っていることからあきらかでしょう。NEC の場合、以前からのソフトウェア資産の継承のためにシリンダ、ヘッド、セクタによる BIOS コールを残さざるを得ないのでは…と想像されます。ちなみに UNIX でもパーティションをシリンダ単位にしています。

■ 他社製ハード・ディスクを採用しているサード・パーティの現状

55 ボードでは Mode Sense コマンドを発行して、ページ3(Format Device Page)、ページ4(Rigid Disk Drive Geometry Page)から得たシリンダ数、ヘッド数、トラック当たりのセクタ数を SCSI 管理領域に書き込んでいます。また、これらの値を乗算(シリンダ×ヘッド×セクタ/トラック)した結果が Read Capac-

図11 DDR100D の交代セクタの割り付け

	セクタ 0	セクタ 1	セクタ 2	セクタ 3	セクタ 32	セクタ 33	セクタ 34	セクタ 35
ヘッド0									交代 セクタ
ヘッド1									交代 セクタ
ヘッド2									交代 セクタ
ヘッド3									交代 セクタ

ゾーンあたりのトラック数4がヘッド数4と等しいことからシリンダ単位に交代セクタ管理を行っていることと、ゾーンあたりの交代セクタが8(論理セクタ4)であることからトラック中に1交代セクタがあるとみなす。

図12 交代セクタがヘッド数で割り切れない例

	セクタ 0	セクタ 1	セクタ 2	セクタ 3	セクタ N-4	セクタ N-3	セクタ N-2	セクタ N-1	セクタ N
ヘッド0										
ヘッド1										
ヘッド2										
ヘッド3								交代 セクタ	交代 セクタ	交代 セクタ

図11と同様に交代セクタの管理がシリンダ単位だがゾーンあたりのヘッド数4に対し、ゾーンあたりの交代セクタ数が3で、ヘッド数で割り切れない。

ity から得た論理ブロックの総数(論理ブロック・アドレス+1)と一致しているのは前述しました。NEC のハード・ディスクは、パーティションの単位がシリンダであり、純正ドライブ(D3835)の解析でわかるように、少なくともシリンダの倍数がちょうど 1 M バイトになるようになっています。

$$\begin{aligned} & 80(\text{ヘッド数}) \times 25(\text{セクタ/トラック}) \times 512 \\ & = 1024000 \text{ バイト} \end{aligned}$$

つまり、つぎの 2 点の条件を満足しているのが NEC 純正ドライブといえます。

- (1) Mode Sense データで得た諸元が総セクタ数と一致している。
- (2) シリンダの倍数がちょうど 1 M バイトになる。

では、Mode Sense データでこのような値を返さないサード・パーティ製のハード・ディスクは、どのようにしているのでしょうか。サード・パーティ製のハード・ディスクは、このあたりの抜け道として二つの方式を採っています。

一つは、Mode Sense データが純正ドライブと同じ構造の値を返すハード・ディスクを採用することです。

二つ目の方式は、Mode Sense データではなく Read Capacity データから得た論理ブロック総数からシリンダ、ヘッド、トラック当たりのセクタ数を動作可能な値に逆算するやりかたです。

はじめの方式では、おそらくドライブ・メーカーに NEC 向けバージョンの発注依頼が生じますが、後者では、どのようなハード・ディスクでも接続できることから多くのサード・パーティがこの方式を採用しています。この方式は、Read Capacity データから得た論理ブロック総数を固定したセクタ/シリンダで割り、総シリンダ数を求める方法です。当然、割り切れない場合、余りは未使用領域となりますが、ハード・ディスクの諸元に依存しない最善の方法といえます。

SCSI の場合、読み書きは論理ブロック番号を指定する方法しかなく、仮に HD BIOS でシリンダ、ヘッド、セクタ指定の BIOS コールが呼ばれたとして、その値を論理ブロック番号に変換して SCSI BIOS をコールすればよいわけです。ハード・ディスクは渡された論理ブロック番号と、アロケーション長のみが必要であり、シリンダ、ヘッド、セクタなどの緒元が実際のものと違っていても関知できないので、ごまかすことができます。



互換性を阻害する要因

他機種用の SCSI ハード・ディスクが動作しない論理的理由は、イニシエータ側が Inquiry データにプロテクトを掛けている、といったもの以外にも、いろいろあります。

(1) ディスコネクト/リコネクトの未サポート

ディスコネクト/リコネクトが必要なシステムでハード・ディスクにその機能がないことがあります。

(2) Mode Select ページの未サポート

また、システムの立ち上げ処理において、Mode Sense コマンドで得たページの内容がシステムに不都合なため、これを更新しようと Mode Select コマンドを発行したが、変更できないページであったりする場合があります。

変更可能かどうかは Mode Sense コマンドでのページでわかるはずなので、Mode Select で変更しようとする自体、おかしい話です。もっとも、システ

ムの動きにかかわる機能的な条件が合致していなければ動作させなくても当然といえるでしょう。

(3) プロトコル上の問題

またプロトコル上に問題のある例ですが、非同期から同期転送モードに移るためのメッセージ・イン/アウト・フェーズでのメッセージのやり取りで、イニシエータまたはターゲットが同期転送を了解、もしくは取りやめのメッセージが組み込まれていないものまであります。

(4) コネクタ仕様の不一致

物理的な原因の代表にコネクタ仕様の違いがあります。米国ではフル・ピッチ(0.1 インチ・ピッチ)が多く、日本ではハーフ・ピッチ(0.05 インチピッチ)のものが増えています。当然、コネクタ形状が異なるので、そのままでは接続することはできません。変換ケーブルは自作するまでもなく秋葉原で入手できます。

PC-9801-92 ボード

このことは、純正ドライブ(D3835)自身の Mode Sense データのヘッド数が 80(ディスク枚数 40)という想像もできない値を返すことから、ハード・ディスクか BIOS ROM のどちらでごまかすかの違いこそあれ、妥当性があるといえます。

余談になりますが、シリンダ、ヘッド、セクタ番号から論理ブロック番号の求め方を以下に示します。

CYL_ADRS : シリンダ・アドレス

HED_ADRS : ヘッド・アドレス

SEC_ADRS : セクタ・アドレス

HED_CYLN : シリンダ当たりのヘッド数

SEC_TRCK : トラック当たりのセクタ数

とすると

論理ブロック番号 = (CYL_ADRS × HED_CYLN + HED_ADRS) × SEC_TRCK + SEC_ADRS

しかしながら、サポートの都合からか、サード・パーティの SCSI ボードが純正、自社製以外のハード・ディスクでも動作できると明言しているところは少なく、その意味では本家とかわりばえしませんが、実際には汎用化はおおいに進んでおり、他社製ドライブを接続してみると動作するボードもけっこうあります。また、同期転送モードをサポートしているボードもいくつか発表されています。サポートの意味からは、汎用 SCSI ボードは割りがあわないでしょうが、ユーザの立場からは、もっと多く市場に出回ってほしいものです。

SCSI の本場アメリカに目を向けましょう。IBM PC 用に汎用 SCSI ボードは、もはや当たり前です。もしも汎用 SCSI ボードで動作しないハード・ディスクがあれば、そのハード・ディスクのプロトコルが疑われるほどに汎用性が高いそうです。Macintosh の場合、付属のインストーラ(ホスト・コンピュータにハード・ディスクを認識させるソフト)は純正ハード・ディスクとして登録していないものは無視してしまうので、ハード・ディスクのサード・パーティが独自にインストーラを用意して製品としています。サード・パーティ製または汎用のインストーラ(日本で 2 万円前後)の中には、純正のインストーラで組み込んだときより高速にアクセスできるものがあります。

筆者も以前に汎用のインストーラで物理フォーマットをしたときに、ホスト・コンピュータとハード・ディスクの組み合わせで理想的なスピードを実現できるインタリーブ値で自動的に物理フォーマットされたのには驚かされました。

'93 年 7 月に発表された NEC SCSI I/F ボード PC-9801-92 については、はじめ「Inquiry コマンドのプロテクトをはずした」と聞いていました。しかしながら、55 ボードの項で述べた Mode Sense データの値を諸元領域(0460h)に書き込んでいる問題もあるので、単純に他社製ハード・ディスクに対し、Inquiry データの“NEC”を確認しないだけでは、Mode Sense データが NEC 製ハード・ディスクと同一形式(物理諸元が論理諸元と一致)でないかぎり、整合はとれません。また、サード・パーティのボードが採用している Mode Sense データを使わない論理諸元形式(総論理ブロック数を固定したセクタ/シリンダで割り、総シリンダ数を求める形式)では、55 ボードでインストール済みの NEC 製ハード・ディスクのデータが使えないことになります。以上の理由から、Inquiry データの使われ方に着目して検証します。

■ 55 ボードの問題点

PC-9801-92 ボードの検証の前に、すでに検証した 55 ボードと、他社製ハード・ディスクとの問題点を整理すると、

(1) Mode Sense データから セクタ長、セクタ/トラック、ヘッド、シリンダ数を得て、そのままの値を諸元領域(0460h)に書き込んでいる。書き込まれた値を乗算した値が総セクタ数と一致している。そのため、物理セクタ・サイズと論理セクタ・サイズの違うハード・ディスクや、トラック中に交代セクタが Mode Sense データ中に書き込まれているハード・ディスクなどとの互換がとれない。

(2) Mode Sense データ(ページ・コード 3Fh: 全ページ)の構造を図 13 の固定した構造と見なし、セクタ長、セクタ/トラック、ヘッド、シリンダ数を固定されたバイト・ポインタから得ている。

このため、これらの書式でデータを返さない、あるいはページ 1 とページ 3 の間で Page 2: Disconnect-Reconnect を返すタイプのハード・ディスクとの互換がとれない。

以上の理由から、Inquiry データのプロテクトをはずしただけではないことが想像されます。

■ PC-9801-92 ボードの検証

このボードを検証するために、NEC 純正ドライブのほかに、極端な値を返す(または返さない)ドライブでシステムを立ち上げたところ、NEC 純正ドライブ以外のドライブでも 55 ボードと違い、永久ループすることなく立ち上がりました。

Mode Sense データとの違いを 0460h から始まる諸元テーブルに書き込まれた値から確認してみます。

検証したドライブ(各数値は Mode Sense, Read Capacity から得た値)

① NEC 純正(セクタ長=512, セクタ/トラック=23h, ヘッド=0Fh, シリンダ数=4C5h)

総セクタ数=9C801h

② ドライブ A(セクタ長=512, セクタ/トラック=20h, ヘッド=40h, シリンダ数=1EAh)

総セクタ数=F5000h

③ ドライブ B(セクタ長=512, Mode Sense データの Page 3, 4 を返さないドライブ)

総セクタ数=48005h

なお、このドライブは Block Descriptor(8 バイト)中の Number of Blocks(総セクタ数)の数も 0 となっています。

<システム立ち上がり後の諸元テーブルの値>

① 23 0F C4 94(セクタ長=512, セクタ/トラック=23h, ヘッド=0Fh, シリンダ数=4C4h+1)

② 20 08 4F 1F(セクタ長=512, セクタ/トラック=20h, ヘッド=08h, シリンダ数=F4Fh+1)

③ 20 08 7F 94(セクタ長=512, セクタ/トラック=20h, ヘッド=08h, シリンダ数=47Fh+1)

<諸元テーブルの値から総セクタ数を算出>

① $23h \times 0Fh \times 4C5h = 9C801h$

② $20h \times 08h \times F50h = F5000h$

③ $20h \times 08h \times 480h = 48000h$

<結果の説明>

(1) ①の NEC 純正ドライブは、Mode Sense データの値がそのまま諸元テーブルに書き込まれており、それらの値の乗算結果が完全に総セクタ数と一致している。

(2) ②のドライブ A は、ヘッド数が 40h から 08h に、シリンダ数が 1EAh から F50h に変更され、総セクタ数に変化はない。

(3) ③のドライブ B は、Mode Sense データが返送されないにもかかわらずセクタ/トラック、ヘッド数が②と同じ数値で書かれており、総セクタ数も Read Capacity コマンドで得た数値より、わずか 5 セクタ少ない値になった。

■ 考察 92 ボードは自/他社製を判断して対処する

これらの検証結果から、92 ボードが Inquiry データの“NEC”を見なくなったのではないことがわかります。つまり、Inquiry データの“NEC”を使って自社製、他社製の判断を行い、以下の動作をします。

▶ NEC 製であれば従来どおり Mode Sense データの値をそのまま諸元テーブルに書き込む。

▶ NEC 製でなければセクタ/トラック、ヘッド数を固定した値(セクタ/トラック=20h, ヘッド数=08h)とし、シリンダ数は Read Capacity で得た論理ブロック総数(図14)をセクタ/シリンダ(20h×08h)で割って総シリンダ数とし、諸元テーブルに書き込む。このことは、Mode Sense データの Page 3, 4 ばかりでなく、Block Descriptor 中の Number of Blocks も返さないドライブ B の結果(3)から証明できる。

図13 98 用 NEC ドライブの Mode Sense データ構造

Parameter Header(4h)	
Block Descriptor(8h)	セクタ長を得る
Page1:Read-Write Error Recovery(8h)	
Page3:Format Device(24h)	セクタ/トラックを得る
Page4:Rigid Disk Geometry(19h)	ヘッド、シリンダ数を得る
Page n	

図14 Read Capacity データ・フォーマット

バイト	0	1	2	3	4	5	6	7
	論理ブロック・アドレス				論理ブロック長			

*論理ブロック・アドレス+1が論理ブロック総数となる

特筆すべきことは、これほどのファームの変更をしたにもかかわらず、従来の 55 ボードのユーザがハード・ディスクを再インストールをすることなく 92 ボードが使えることです。先に述べた「NEC 製であれば従来どおりの方式」を採用することで互換性を保ち、かつ他社製ハード・ディスクも接続可能となっています。

参考文献

- 1) SCSI-2 X3T9.2/86-109, ANSI 規格書
- 2) 「SCSI インターフェーステクニカルブック」, 関コーラル

まき・よしもと

4.2

AT 互換機と ASPI プログラミング

鈴木祥夫/吉野 誠

AT 互換機の世界では、ISA および EISA というシステム・バスの標準が確立しているため、この標準バスのスロットに差す SCSI ホスト・アダプタ・ボードが市場にたくさん出まわっている。最初は、このアダプタ・ボードごとにデバイス・ドライバが作られていたが、アダプタ・ボードの違いを吸収する ASPI という切り口が Adaptec 社によって提案された。その結果、ボードとデバイス・ドライバの両側から、この ASPI に準拠した製品が登場し、AT 互換機では、SCSI による周辺機器の接続がグンと楽になった。ここでは、まず、この ASPI 仕様について説明してから ASPI のプログラミング例を示す。 (編集部)

SCSI で接続された機器を IBM AT および互換機 (以下、AT 互換機と表記する) 上からコントロールするには、デバイス・ドライバが必要です。しかし、デバイス・ドライバを自分で開発しようとするパソコン上に搭載された SCSI ホスト・アダプタの種類にしたがって作成する必要がありますから、別の SCSI ホスト・アダプタを使用するとふたたび同じような機能のドライバを作る手間が必要でした。

● なぜ ASPI か

Adaptec 社の提唱した ASPI (Advanced SCSI Programming Interface) は、その SCSI ホスト・アダプタのメーカー別に対応する部分を共通のインターフェースによりアクセスできるようにするものです。これによって、(AT 互換機の場合は) SCSI ホスト・アダプタの種類に関係なく、ドライバは ASPI 対応のものだけ開発すればいいことになりました。

1

SCSI と ASPI の関係

現在、SCSI は、ハード・ディスクや CD-ROM などの大容量の記憶装置とコンピュータを接続する標準インターフェースとして、AT 互換機や、ワークステーションなどで広く使われています。記憶装置以外にも、イメージ・プリンタなどの高速データ転送が必要な機器でも使われています。

SCSI は SCSI ホスト・アダプタと各種機器をバスを介して接続するためのハード仕様やコマンドなどは定義されていますが、残念ながらホスト・アダプタとドライバ・ソフトの間の仕様は定義されていません。

しかも SCSI ホスト・アダプタ・ボードは、現在、世界中でいろいろなメーカーで製造され、それらのボード上にはさまざまなコントローラ・チップが搭載されていますから、それらをコントロールするデバイス・ドライバを作成するとなると、SCSI ホスト・アダプタ別に異なるプログラムをいちいち作成しなければなりませんでした。

せっかく各種機器を SCSI という共通なインターフェースで接続するにしてもコンピュータに搭載された SCSI ホスト・アダプタの種類により別々のデバイス・ドライバを用意する必要があったわけです。

そこでデバイス・ドライバの機能を大きく二つに分割し、ホスト・アダプタにアクセスする部分を独立させることにより、ホスト・アダプタの違いをすべてその部分に吸収させてしまおうとして作られたのが ASPI です。

■ ASPI マネージャ

図15の左側のように、それまでのデバイス・ドライバが、直接ホスト・アダプタをコントロールしていたのに対し、ASPIではデバイス・ドライバとホスト・アダプタの間にASPI マネージャと呼ばれるデバイス・ドライバが新たに加わっています。

このASPI マネージャにホスト・アダプタを直接制御する機能を吸収してしまい、機器のデバイス・ドライバからはその機能だけをfarコールで間接的に呼び出します。この関数のエントリ・ポイントを得る方法と、関数の引き数の形が、ASPIの仕様として取り決められたのです。

関数を呼び出すエントリ・ポイントは、一つだけしか用意されていません。その関数の引き数で渡されるテーブルの中にコマンド番号を指定することで機能が分けられます。現在定義されている機能には、

- ▷ ホスト・アダプタの問い合わせ
- ▷ デバイス・タイプの獲得
- ▷ SCSI コマンドの実行/中止
- ▷ デバイスのリセット
- ▷ ホスト・アダプタのパラメータ・セット
- ▷ ディスク・ドライブの情報獲得

などがあります。このテーブルはSRB(SCSI リクエスト・ブロック、表2：p.96)と呼ばれます。

機能のうち、“SCSI コマンド実行”では、SCSI コマンドをそのままテーブルの一部に記述できます。つまり機器をコントロールするためのコマンドをASPIで

新たに定義していないので、SCSI コマンド仕様をこれまでどおりに使うことができます。

しかもSCSI コマンドの実行後に機器から発生するチェック・コンディションに対してのセンス・データの要求という決まった処理フローもASPI マネージャが担当しているので、これらはASPI 内部で自動的に行われ、結果がテーブルにセットされます。

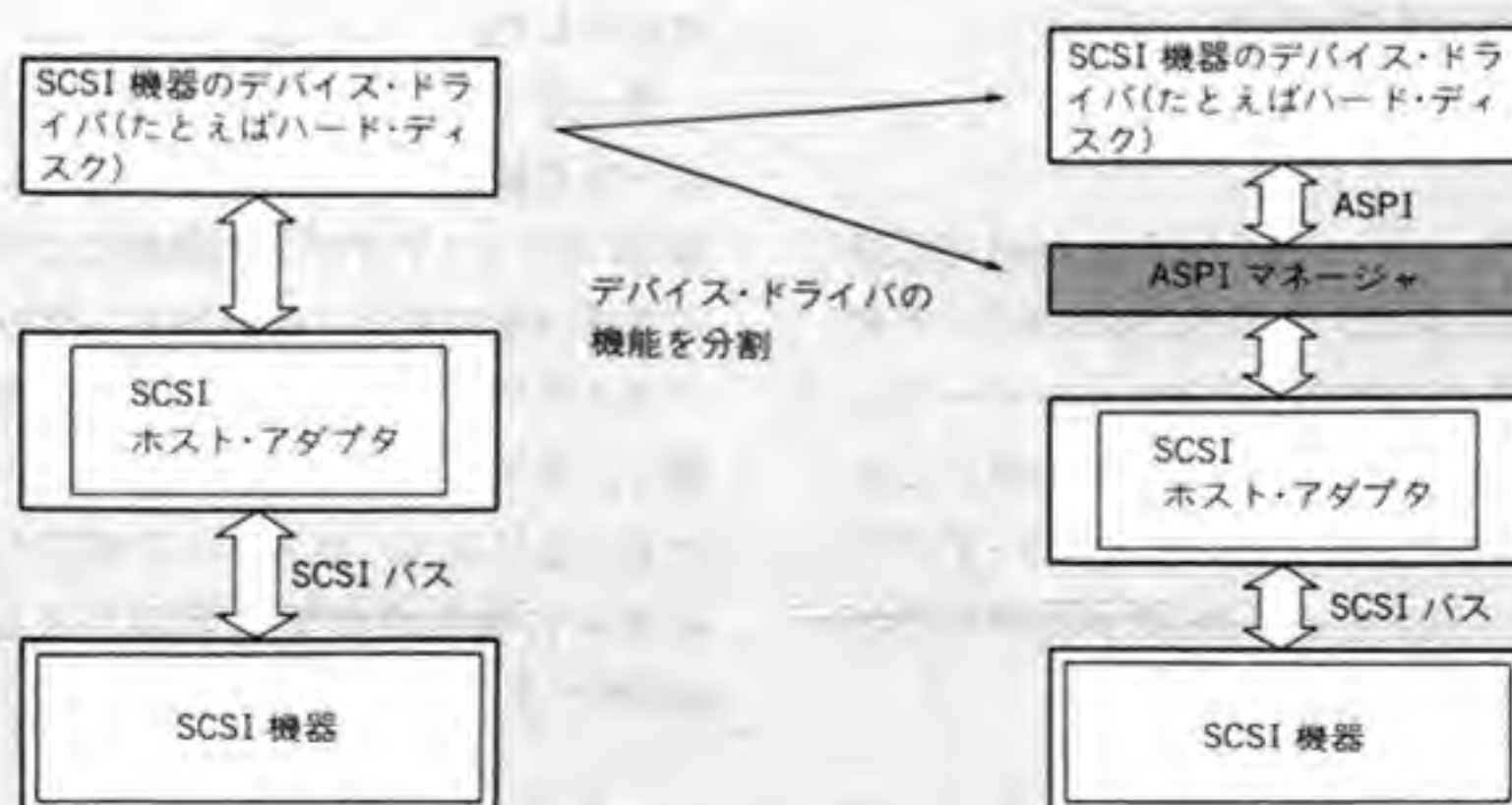
またASPI マネージャへの要求は、処理の完了リターンで戻るのではなく、要求を受け付けるとすぐに戻ってきます。そのため処理の終了を確認する方法は、要求したときに用いたテーブルの中にあるステータスを監視するか、処理終了時に呼ばれるユーザ定義関数を定義して処理する方法のどちらかで行います。要求し、完了するまでの間、要求するときに用いたテーブルは静的なエリアで、かつ変更できません。

各デバイス・ドライバからの制御は、すべて一つのASPI マネージャを通してからSCSI ホスト・アダプタ、SCSI バス、各デバイスへ流れます。しかも一つのASPI マネージャはリエントラント構造で作成されていますから、図16のように複数のデバイス・ドライバからの制御を受け付けることができます。

ASPIに対応しているメーカーは、Adaptec社以外に、Always Technology Corp., BusLogic Inc.などで、各社はだいたいアダプタ・ボードと共にASPI マネージャを提供しています。

このようにAT互換機の世界ではASPIに対応したホスト・アダプタが数多く出まわってきていますが、

図15 従来のSCSI デバイス・ドライバとASPIの違い



NEC の PC-98 の世界で ASPI を使おうとするとまだ多くの問題があるため、いまのところ対応できていません。

現在 Adaptec 社でも 98 マシン用の SCSI ホスト・アダプタと ASPI マネージャを開発中のようですが、ハード・ディスクのフォーマットが NEC 独自のフォーマットであるため、ハード・ディスクの対応には専用のドライバが必要となってしまいます。

また ASPI に対応したハード・ディスク以外のデバイス・ドライバやユーティリティ・ソフトも AT 互換機で使うことを前提とした作り方をしているものがあり、そのまま使えない場合もあります。

■ ASPI を使う利点

ASPI を使うことにより各社から提供されている SCSI ホスト・アダプタに対応した各種機器のデバイス・ドライバを作成する手間が省けるだけではありません。

パッケージ開発者にも CD-ROM やハード・ディスクなどの各種機器に対するユーティリティを SCSI ホスト・アダプタの種類を気にすることなく開発することができる利点があります。

また、通常のデバイス・ドライバの作成時のように I/O 命令で直接ボードをアクセスする必要がないため、機器のコントロールに専念したプログラミングがスピーディにできます。

現在、ASPI に対応したユーティリティ・ソフトや、

ボード・メーカー以外のデバイス・ドライバも発売されています。

Adaptec の EZ-SCSI (コラム: p. 94~参照) や Corel の Corel SCSI は、SCSI に接続されているホスト・アダプタや接続されているデバイスの種類による違いを自動的に選別し、複雑な SCSI ドライバやデバイス・ドライバのインストールを簡単に行えるようにしたツール・ソフトです。

SCSI のユーティリティ・ソフトには、これら以外にも、ハード・ディスク・フォーマッタやハード・ディスクとストリーマのバックアップ・ユーティリティ、フォト CD ビューアなど、さまざまなものが登場しています。

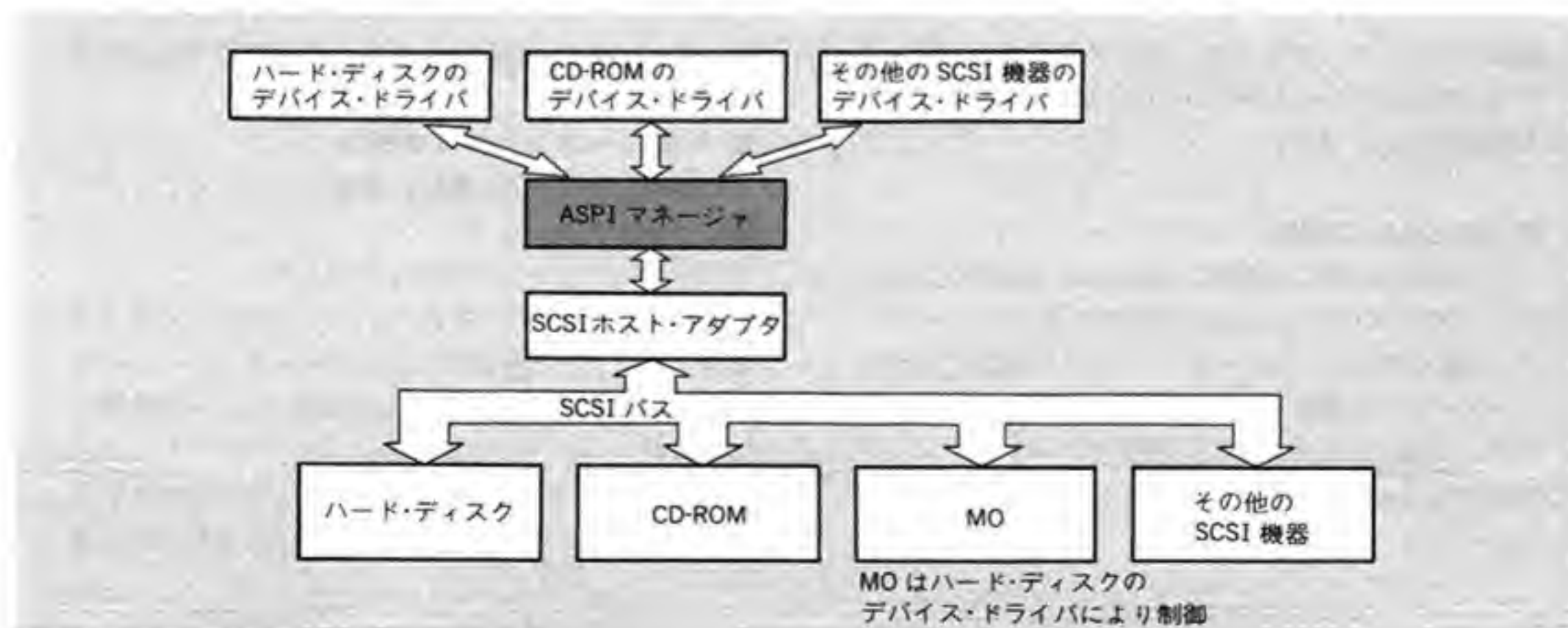
2 ASPI によるプログラミング

ASPI を使うには、最初にその SCSI ホスト・アダプタ用の ASPI マネージャがインストールされていることが前提となります。

ASPI でのデバイス・ドライバの作成は、つぎの順で行います。

- (1) ASPI のエントリ・ポイントを獲得する
- (2) ホスト・アダプタへの問い合わせを行う
- (3) デバイス・タイプを獲得する
- (4) SCSI コマンドを実行する

図16 複数のデバイスを一つの ASPI マネージャで制御できる



エントリ・ポイントはデバイス・ドライバの中で、最初に1回だけ獲得します。そして、獲得したエントリ・ポイントを使って ASPI コマンドを発行し、SCSI ドライバをコントロールするプログラムを作成します。

表1 (p. 96) が ASPI で定義されているコマンド・コードと機能です。表からわかるように、ASPI コマンドは基本的には7個です。このうちコマンド・コード 00h の機能は「ホスト・アダプタのタイプは何で、何個ついているか、ドライバは何か」を問い合わせるコマンドです。

■ DOS 版での使い方

つぎに DOS 版による ASPI の使い方を説明します。

(1) エントリ・ポイントの獲得

エントリ・ポイントを獲得するには、MS-DOS のシステム・コールの INT 21h のファンクション・コールを使います。

AX レジスタに 3D00h、DS レジスタと DX レジスタに、文字列 "SCSIMGR\$" のセグメント・アドレスとオフセット・アドレスをそれぞれセットします。文字列の最後に 00h を忘れないようにします。



Adaptec 社のデベロッパーズ・キットの概要

ASPI による開発の手助けとして Adaptec 社からデベロッパーズ・キットが発売されています。これには、ASPI によるアクセス手順が、DOS や Windows, NetWare, OS/2 それぞれにわけて説明されたプログラミング・マニュアルと、開発の具体例として DOS でのデバイス・タイプの読み込みと、Windows での SCSI バスに接続された機器を調べるスキャン・プログラムがソース・コードと共に入っています。

■ デベロッパーズ・キットの内容

デベロッパーズ・キットに含まれるフロッピー・ディスクには、DOS 版と Windows 版のサンプル・ソースが含まれています。これらのソースには、ASPI コマンド、SCSI 機器の各コマンド(共通コマンド、ダイレクト・アクセス・デバイス用コマンド、CD-ROM デバイス用コマンドなど) がインクルード・ファイルとして含まれています。

その他、ASPI マネージャやデバイス・ドライバを自動的にセットアップしたり、ディスクをフォーマットする EZ-SCSI ツールや CD-ROM のユーティリティも同梱されています。

■ EZ-SCSI の概要

EZ-SCSI は AT 互換機に、Adaptec 社製の SCSI ホスト・アダプタや AIC-6260/6360 プロトコル・チップ、AIC-7770 コントローラ・チップを使用するホスト・アダプタを搭載しているとき、DOS や Windows の深い知識がなくても SCSI 機器のハード・ディスク、CD-ROM を容易にインストールできるようにしたツールです。

ツールを起動するとコンピュータに現在搭載されている SCSI ホスト・アダプタを自動的に検出し、その種類を表示します(写真)。

さらに、そのホスト・アダプタに接続されている SCSI 機器をサーチし、検出したすべての機器名称とデバイス ID 番号を表示します。

必要ならばそれぞれの機器が SCSI-1、SCSI-2 のどちらで接続されているか、同期転送をサポートしているか、などの表示も行います。

Windows 用の ASPI のインストールを行い、CD-ROM が接続されているときは、ベンダ名を自動判定してドライバをインストールします。

EZ-SCSI はオーディオとデータの両方の動作モードに対応した、ベンダ別のドライバがインストールされています。ベンダは、チノン、DEC、DENON、日立、LMS、NEC、Panasonic、Sony、Texel、東芝の各機種に対応しています。

各ドライバをインストール後、それらの情報を config.sys と autoexec.bat のファイルに書き込みます。

■ 付属ユーティリティの紹介

つぎのような SCSI 機器の各種ユーティリティが付属しています。

(1) ハード・ディスクのフォーマット

SCSI ハード・ディスク、リムーバブル・メディア、光磁気ディスクの機器に対するフォーマット・ユーティリティとして afdisk と scsifmt の二つが付属しています。

afdisk はハイレベルのフォーマット・ユーティリティで、ホスト・アダプタ BIOS によってマウントさ

そして INT 21h のソフト割り込みによりハンドルをオープンします。

リターン後、キャリー・フラグがセットされている場合は異常終了なので、ASPI マネージャがインストールされていないことが考えられます。

キャリー・フラグがセットされない場合が正常終了で、AX レジスタにハンドル番号が返されます。

つぎに、AX レジスタに 4402h をセット、BX レジスタにオープンで得たハンドル番号を、CX レジスタに 4

を、DS レジスタと DX レジスタに ASPI エントリ・ポイントを格納するためのセグメント・アドレスとオフセット・アドレスをセットし、同様に INT 21h のソフト割り込みで 4 バイトの ASPI エントリ・ポイントを得ます。

エントリ・ポイントを得た後は、ハンドルをクローズするために、AX レジスタに 3Eh を、BX レジスタにハンドル番号をセットし、INT 21h のソフト割り込みを行います。

れない SCSI ディスクをパーティションして使う必要があるときに使います。これにより、一つのハード・ディスクを複数の論理ドライブにして使いたいときに任意のパーティションで区切ることができます。

もうひとつはロー・レベル・フォーマット・ユーティリティの scsifmt で、これは物理的にフォーマットします。

これらのツールによりフォーマットするとディスク上のデータがすべて消えますので注意が必要です。

(2) SCSI の機器情報表示

このツールは Windows 上で動作し、システムにインストールされているすべての SCSI 機器の情報を表

示します。

(3) CD-PLAYER の実行

このツールはキーボードやマウス・インターフェースによりオーディオ CD に対する、Play, Pause, Eject, Stop などの操作や、時間表示、トラック番号の表示を行います。

(4) フォト CD のビューア

このツールは Windows 上で動作し、コダックのフォト CD を見るためのツールです。画像のインデックス表示、時間で順に表示するスライド表示、回転表示を行います。



(a) SCSI デバイスのところをクリアすると...



(b) その SCSI デバイスの緒元がオーバーラップ・ウィンドウに表示される

写真 EZ-SCSI のインストール・ツールの例

表1 ASPI コマンド一覧

コマンド・コード	機 能
00h	ホスト・アダプタの問い合わせ
01h	機器の種別を表すデバイス・タイプの獲得
02h	SCSI コマンドの実行
03h	SCSI コマンドの中止
04h	デバイスのリセット
05h	ホスト・アダプタのパラメータのセット
06h	ディスク・ドライブの情報の獲得
07h~7Fh	将来の拡張用
80h~FFh	ベンダ用

表4 デバイス・タイプの獲得

オフセット	内 容	R/W
00h	ASPI コマンドのコード1をセット	W
01h	コマンド実行のステータス	R
02h	ホスト・アダプタの番号	W
03h	0 をセット	W
04h	0 をセット	—
08h	機器のターゲット ID	W
09h	ロジカル・ユニット番号	W
0Ah	機器のデバイス・タイプ	R

あとは獲得したエントリ・ポイントを使ってファンクション・コールで ASPI マネージャにアクセスします。

ファンクション・コールするとき SCSI リクエスト・ブロック (以下 SRB) と呼ばれるテーブルに表1で述べたコマンド・コードとコマンドに対応した情報をセットし、そのアドレスをスタックにセットします。

SRB の先頭から 8 バイトは共通のフォーマットになっています (表2)。

先頭が表1のコマンド・コードです。ホスト・アダプタ番号は0から始まり、複数のホスト・アダプタが実装されているときに目的のホスト・アダプタを選択するためにセットします。

コマンド・コードのところには ASPI コマンドを書き込みます。

ステータス・バイトはファンクション・コールをする前に0にします。ファンクション・コール後にリクエストの動作モードを表します。

表2 SCSI リクエスト・コマンド: SRB の基本構造

先頭からのオフセット	内 容	R/W
00h	ASPI コマンド・コード	W
01h	ステータス・バイト	R
02h	ホスト・アダプタ番号	W
03h	SCSI リクエスト・フラグ	W
04h (4 バイト)	拡張用に予約	—

表3 ホスト・アダプタの問い合わせ

オフセット	内 容	R/W
00h	ASPI コマンドのコード0をセット	W
01h	コマンド実行のステータス	R
02h	ホスト・アダプタの番号	W
03h	0 をセット	W
04h	0 をセット	—
08h	ホスト・アダプタの数	R
09h	ホスト・アダプタのターゲット ID	R
0Ah (16 バイト)	SCSI ドライバの ID 文字列	R
1Ah (16 バイト)	SCSI ホスト・アダプタの ID 文字列	R
2Ah (16 バイト)	ホスト・アダプタの固有の情報	R

(2) ホスト・アダプタの問い合わせ

実装されているホスト・アダプタの数やホスト・アダプタの情報を獲得するコマンドです。

SRB のホスト・アダプタ番号を0にするとホスト・アダプタの数を得ることができ、目的のホスト・アダプタ番号を指定するとホスト・アダプタの種別のための文字列を得ることができます。

SRB の詳細は表3のとおりです。

SRB に情報をセットし、ファンクション・コールを行います。処理の完了を待たずにリターンしてきます。そのため処理が終了したかどうかはコマンド実行のステータスで監視します。ステータスの値が0の間は処理が進行中で、正常に終了したときには01hになるので、それから各情報の読み込みが可能になります。

(3) デバイス・タイプの獲得

SCSI に接続された機器の情報を獲得するコマンドです。目的の機器のターゲット ID とロジカル・ユニット番号 (LUN) をセットし、ファンクション・コールを

表 5 SCSI コマンドの実行

オフセット	内 容	R/W	オフセット	内 容	R/W
00h	ASPI コマンドのコード 2 をセット	W	11h	(セグメント)	W
01h	コマンド実行のステータス	R	13h	SCSI コマンドのリンク時, SRB のつぎへのポインタ(オフセット)	W
02h	ホスト・アダプタの番号	W	15h	(セグメント)	W
03h	SCSI リクエスト・フラグ	W	17h	SCSI コマンドの長さのバイト数(M)	W
04h	0 をセット	—	18h	ホスト・アダプタのステータス	R
08h	機器のターゲット ID	W	19h	ターゲットのステータス	R
09h	ロジカル・ユニット番号	W	1Ah	ポスト・ルーチン・アドレス(オフセット)	W
0Ah	転送するデータのバイト数 転送データがないときは 0 をセット	W	1Ch	(セグメント)	W
0Eh	SCSI コマンドのセンス・データの最大 バイト数(N)	W	1Eh	ASPI ワーク・エリアに予約済み	—
0Fh	転送データのバッファ・アドレス (オフセット)	W	40h	SCSI コマンドを記述するエリア, 長さは M	W
			40h + M	センス・データを格納するエリア 長さは N	R

行くと機器のデバイス・タイプが返されます。デバイス・タイプは、SCSI 仕様で規定されている番号で、00h がディスク、01h がテープ、02h がプリンタ、05h が CD-ROM、06h がスキャナなどを表します(表 4)。

このコマンドを使って、ターゲット ID を順にスキャンすれば接続している機器の種類を確認することができます。

(4) SCSI コマンドの実行

実際の機器のコントロールは、この ASPI コマンド実行のコマンドを使い、SCSI コマンドの発行により行います(表 5)。

このコマンドの SRB は今までの SRB と比べて、設定する項目がかなり増えますが、大きくわけるとつぎの三つの部分、

- ① ASPI のコマンド部
- ② SCSI のコマンド部
- ③ センス・データ部

から構成されています。

プログラム側の転送データ・バッファのアドレスや大きさ、さらに処理が終了したときに呼ばれる関数アドレス(ポスト・ルーチン・アドレス)を ASPI コマンド部に設定することができます。

ポスト・ルーチン機能を使うと処理が終了したかどうかをステータスで監視する必要がなくなり、プログラムは他の処理を行うことができます。

ポスト・ルーチンとは、処理が終了したときに ASPI

マネージャから呼ばれるユーザ定義関数をいいます。

ただし、ポスト・ルーチンの中では、後述する ASPI コマンドの Abort リクエストは発行できません。

また、割り込み処理の中で呼ばれることが多いので、処理時間なるべく短くなるように記述します。

SCSI コマンド部には、SCSI で定義されているデバイス・タイプ別のコマンド・ディスクリプション・ブロック(CDB)をそのまま記述します。SCSI コマンドは可変長なので、その長さをバイト数でセットします。

SCSI コマンドについては第 2 章や SCSI の仕様書を参考にしてください。

センス・データ部は、SCSI コマンド実行後に、チェック・コンディションが検出されたときに、ASPI マネージャがターゲットに対し、自動的にリクエスト・センスを発行し、センス・データを取り出し格納します。

そのほか、ASPI コマンド部の SCSI リクエスト・フラグには、つぎの項目をビット単位で指定します。

- ▷ 処理後に呼ぶ関数(ポスト・ルーチン・アドレス)を定義したいときに 0 ビットに 1 をセットする。
- ▷ SCSI コマンドのリンクを使用するときに 1 ビット目に 1 をセットする。

ただし、この機能は SCSI ホスト・アダプタや、ターゲットが対応していない場合があり、正常に動作しないことがあるので使わないほうがいいでしょう。

▷ データの転送方向を 3~4 ビット目にセットする。

SCSI コマンドに決定される方向のとき: 00

ターゲットからホスト・アダプタへ転送されるとき: 01

ホスト・アダプタからターゲットへ転送するとき：10
データ転送がないとき：11

そのほかの定義されている ASPI コマンドを以下に説明します。

(4) SCSI I/O リクエスト中止

現在実行中の ASPI リクエストを中止したいときに発行します。実行したときの SRB のアドレスを指定して、この中止リクエストを発行すると中止の処理が始まります。

このリクエストそのものは、必ずステータスが1のエラーなしで終了しますが、実際に実行している ASPI リクエストの中止処理が完了したかどうかは実行している ASPI リクエストの SRB のステータスを監視してチェックします (表 6)。

(5) SCSI デバイスのリセット

接続されている機器に SCSI コマンドのデバイス・

表 6 SCSI I/O リクエスト中止

オフセット	内 容	R/W
00h	ASPI コマンドのコード 3 をセット	W
01h	コマンド実行のステータス	R
02h	ホスト・アダプタの番号	W
03h	0 をセット	W
04h	0 をセット	—
08h	SRB ポインタ(オフセット)	W
0Ah	(セグメント)	W

表 7 SCSI デバイスのリセット

オフセット	内 容	R/W
00h	ASPI コマンドのコード 4 をセット	W
01h	コマンド実行のステータス	R
02h	ホスト・アダプタの番号	W
03h	SCSI リクエスト・フラグ	W
04h	0 をセット	—
08h	機器のターゲット ID	W
09h	ロジカル・ユニット番号	W
0Ah	予約済み	—
18h	ホスト・アダプタのステータス	R
19h	ターゲットのステータス	R
1Ah	ポスト・ルーチン・アドレス (オフセット)	W
1Ch	(セグメント)	W
1Eh(2バイト)	ASPI ワーク・エリアに予約済み	—

リセットを発行します (表 7)。

(6) ホスト・アダプタ・パラメータ・セット

ホスト・アダプタに固有のパラメータをセットするコマンドです (表 8)。セットする内容はホスト・アダプタの種類により異なりますので、ホスト・アダプタに付属している説明書を参考にする必要があります。

(7) ディスク・ドライブの情報を獲得

このコマンドにより指定されたディスク・ドライブ機器がすでに BIOS/DOS で管理下にあるかどうかを調べることができます (表 9)。

このコマンドはつねに完了ステータスが1か、80hのどちらかでリターンします。80hでリターンしたときは ASPI がこのコマンドをサポートしていないときです。

正常終了するとドライブ・フラグの最下位の2ビットにつぎの情報を返します。

▷指定されたドライブは INT 13h によりアクセスできない：00

▷指定されたドライブは DOS の管理下にあり INT 13h によりアクセスできる：01

▷指定されたドライブは DOS の管理下ではないが INT 13h によりアクセスできる：10、アクセスできるときは INT 13h ドライブ番号のエリアにドライブ番号を返す。

■ Windows 版での使い方

Windows 上で ASPI を使用するときは、DOS と同じように直接呼ぶことはできません。DOS はリアル・モードで動作しますが、最近の Windows はプロテクト・モードで動作するので、直接ポインタやバッファのアドレスをセットできません。

そのため、Adaptec の ASPI マネージャには Windows 用として、DLL(Dynamic Link Library)の形の winaspi.dll も提供されているので、Windows のプログラムから簡単にアクセスすることが可能です。

この winaspi.dll にはつぎの二つの関数が用意されています。

- ・ GetASPISupportInfo
- ・ SendASPICommand

GetASPISupportInfo はこれから ASPI を使うことを宣言します。この関数には引き数がなく、リターン

値でホスト・アダプタの数と、ASPI マネージャの状態を HIBYTE と LOBYTE に分けて返します。

SendASPICommand は DOS の ASPI の部で述べた SRB へのポインタを引き数とした関数です。

この関数は DOS 用でインストールされている ASPI マネージャとデータのやり取りを行う形で動作します。

表 8 ホスト・アダプタ・パラメータ・セット

オフセット	内 容	R/W
00h	ASPI コマンドのコード 5 をセット	W
01h	コマンド実行のステータス	R
02h	ホスト・アダプタの番号	W
03h	0 をセット	W
04h	0 をセット	—
08h(10 バイト)	ホスト・アダプタ固有のパラメータ	W

表 9 ディスク・ドライブの情報を獲得

オフセット	内 容	R/W
00h	ASPI コマンドのコード 6 をセット	W
01h	コマンド実行のステータス	R
02h	ホスト・アダプタの番号	W
03h	0 をセット	W
04h	0 をセット	—
08h	機器のターゲット ID	W
09h	ロジカル・ユニット番号	W
0Ah	ドライブ・フラグ	—
0Bh	INT 13h ドライブ番号	R
0Ch	ヘッド変換値の推奨値	R
0Dh	セクタ変換値の推奨値	R

3 ASPIによるサンプル・プログラム

DOS で ASPI を使用した例のサンプルを稿末に示します。リスト 1 は、ASPI エントリを得る関数と、七つある ASPI コマンドのうち、よく使われる四つのコマンドを C 言語により関数化したものです。

リスト 2 は、リスト 1 の関数を使い、実際に SCSI デバイスへ問い合わせコマンドを発行し、その結果を表示するプログラムです。

C 言語で記述していますので、より理解しやすいと思います。コマンド単位で関数化していますから、どんなデバイスでも使えますが、センス・データの大きさを調整する必要があります。

具体的には SRB 構造体の中の CDB の配列のサイズを変えます。

このソースは MS-C バージョン 6 でコンパイルしています。コンパイルはつぎのようにして行います。

```
> cl /Zp aspicall.c
/Zp のオプションを必ず付けてください。
```

使い方は、デバイス ID 番号を引き数としてコマンドを呼び出します。するとそのデバイスの情報が表示されます。

参考文献

- 1) 「ASPI ソフトウェア デベロッパーズ キット マニュアル」, Adapte

すずき・よしお 日本ノーベル㈱
よしの・まこと ㈱エイアンドティー

リスト1 ASPIコール・サンプル・モジュール ①

```

/*
ASPI-CALL sample module

< List of module >
Get_ASPI_Entry      : ASPIエントリーポイントを取得する。
dsp_ASPI_status      : ASPIステータスに対するメッセージを表示する。
Call_ASPI            : ASPIをコールする。
Host_Adapter_Inquiry (ASPI Command Code = 0)
                    : ホストアダプタに関する情報を得る。
Get_Device_Type      (ASPI Command Code = 1)
                    : SCSI接続装置のタイプを得る。
Scsi_IO_Request       (ASPI Command Code = 2)
                    : SCSI接続装置とのデータ転送を行う。
Reset_SCSI_Device    (ASPI Command Code = 4)
                    : SCSI接続装置をリセットする。

1993 by Japan Novel Corp.

*/

//--- Including files about standard module---
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <memory.h>
#include <dos.h>

//--- Define type ---
typedef unsigned short WORD;
typedef unsigned char BYTE;

//--- Define struct ---
#pragma pack(1) // 'struct' boundary : 1 byte by 1 byte
typedef struct {
    BYTE Command; // [0] Command Code
    BYTE Status; // [1] Status
    BYTE Host_Adapter_Num; // [2] Host Adapter Number
    BYTE Request_Flag; // [3] SCSI Request Flags
    BYTE Exp_Reserve[4]; // [4] Reserved for Expansion
    BYTE Target_ID; // [8] Target ID
    BYTE Target_LUN; // [9] LUN
    unsigned long Data_Len; // [10] Data Allocation Length
    BYTE Sense_Len; // [14] Sense Data Allocation Length
    BYTE far *Data_Bp; // [15] Data Buffer Pointer
    void far *srb; // [19] SRB Link Pointer
    BYTE Cdb_Len; // [23] SCSI CDB Length
    BYTE Host_Adapter_Sta; // [24] Host Adapter Status
    BYTE Target_Status; // [25] Target Status
    void far *Post; // [26] Post Routine Pointer
    BYTE Work[34]; // [30] Reserved for ASPI Workspace
    BYTE Cdb[50]; // [64] SCSI CDB & Sense Data Buffer
} SRB;

typedef struct {
    BYTE HA_Count; // Number of Host Adapters
    BYTE HA_SCSI_ID; // SCSI ID of the host adapter
    BYTE HA_ManagerId[16]; // SCSI Manager ID String
    BYTE HA_Identifier[16]; // Host Adapter ID String
    BYTE HA_Unique[16]; // Host Adapter Unique parameters
} SRB_HA;

typedef struct {
    BYTE sense_code;
    BYTE *sense_str;
} SENSE;

//--- Define ASPI entry point and struct ---
SRB srb;
SRB far *far_srb;
SRB_HA srb_ha_inquiry;
void (far *ASPI_ent)(SRB far *);

#define SENSE_LEN 14 // Default request sense buffer length

/*title
| モジュール名 : Get_ASPI_Entry
| 機 能 : ASPIエントリーポイントを取得する。
| コーディング・シケンス : int Get_ASPI_Entry( void )
| 引 数 : nothing
| 関 数 値 : 0 = 正常終了 / -1 = デバイスドライバ・オブジェクト
|           / -2 = エントリーポイント取得エラー
|           / -3 = デバイスドライバ・ロードエラー
| 構 造 体 : struct srb
| インクルード・ファイル : dos.h / stdio.h
| 作 成 者 : 1993 Japan Novel Corp
| 備 考 :
*/

int Get_ASPI_Entry( void )
{

```

```

char far *SCSI_mgr = "SCSIMGRS";
char buf[40];
union REGS reg;
struct SREGS sreg;

//--- Opening ASPI ---
reg.x.ax = 0x3d00;
sreg.ds = FP_SEG( SCSI_mgr );
reg.x.dx = FP_OFF( SCSI_mgr );
intdosx( &reg, &sreg, &sreg );
if( reg.x.cflag & 1 ) {
    printf( "VnSCSIMGRS Open Error %04x", reg.x.ax );
    return -1;
}

//--- Getting the ASPI entry point ---
reg.x.bx = reg.x.ax;
reg.x.ax = 0x4402;
sreg.ds = (unsigned int)((unsigned long)(&ASPI_ent) >> 16);
reg.x.dx = (unsigned int)((unsigned long)(&ASPI_ent) & 0xffff);
reg.x.cx = 4;
intdosx( &reg, &sreg, &sreg );
if( reg.x.cflag & 1 ) {
    printf( "VnSCSIMGRS Read IOCTL Error %04x", reg.x.ax );
    return -2;
}

//--- Closing ASPI ---
reg.h.ah = 0x3d;
intdosx( &reg, &sreg, &sreg );
if( reg.x.cflag & 1 ) {
    printf( "VnSCSIMGRS Close Error %04x", reg.x.ax );
    return -3;
}

//--- Setting ASPI-CALL address ---
far_srb = &srb;
return 0;
}

/*title
| モジュール名 : dsp_ASPI_status
| 機 能 : ASPIステータスに対するメッセージを表示する。
| コーディング・シケンス : void dsp_ASPI_status( void )
| 引 数 : nothing
| 関 数 値 : nothing
| 構 造 体 : struct srb
| インクルード・ファイル : nothing
| 作 成 者 : 1993 Japan Novel Corp
| 備 考 :
*/

void dsp_ASPI_status( void )
{
    static char *ASPI_Status_str[] = {
        "SCSI request aborted by host", // when the status is 0x2
        "SCSI request completed with error", // when the status is 0x4
        "Invalid SCSI request", // when the status is 0x80
        "Invalid host adapter number", // when the status is 0x81
        "SCSI device not installed", // when the status is 0x82
        "Unknown status", // when the status is otherwise
    };
    WORD i;

    switch( srb.Status ) {
        case 2:
            i = 0; break;
        case 4:
            i = 1; break;
        case 0x80:
            i = 2; break;
        case 0x81:
            i = 3; break;
        case 0x82:
            i = 4; break;
        default:
            i = 5; break;
    }

    printf( "Vn## ASPI_Status = [%s]", ASPI_Status_str[i] );
}

/*title
| モジュール名 : Call_ASPI
| 機 能 : ASPIをコールする。
| コーディング・シケンス : void Call_ASPI( void )
| 引 数 : nothing
| 関 数 値 : nothing
| 構 造 体 : struct srb
| インクルード・ファイル : nothing
| 作 成 者 : 1993 Japan Novel Corp
| 備 考 :

```



```

*/
void Call_ASPI( void )
{
    /*--- Call ASPI start ---
    (*ASPI_ent)( far_srb );

    /*--- Wait for status ---
    while( srb.Status != 0 );
}

/*title
-----
| モジュール名 : Clear_SRB
| 機 能 : SRBをゼロクリアする。
| コーディング・タプル : void Clear_SRB( void )
| 関 数 値 : nothing
| 構 造 体 : struct srb
| インクルードファイル : memory.h
| 作 成 者 : 1993 Japan Novel Corp.
| 備 考 :
|-----
*/
void Clear_SRB( void )
{
    int zero = 0;

    memset( &srb, zero, sizeof(SRB) );
}

/*title
-----
| モジュール名 : Host_Adapter_Inquiry [ASPI Command Code = 0]
| 機 能 : ホストアダプタに関する以下の情報を得る。
|          (1)ホストアダプタの数 (1 byte)
|          (2)ホストアダプタのSCSI ID (1 byte)
|          (3)SCSIマネージャID名 (16 bytes)
|          (4)ホストアダプタID名 (16 bytes)
|          (5)ホストアダプタ特有の記述 (16 bytes)
| コーディング・タプル : void Host_Adapter_Inquiry( BYTE Host_Adapter_Num )
| 引 数 : BYTE Host_Adapter_Num = ホストアダプタID
| 関 数 値 : nothing
| 構 造 体 : struct srb
| インクルードファイル : stdio.h
| 作 成 者 : 1993 Japan Novel Corp.
| 備 考 : このASPIコマンドは必ず0以外のステータスを返す。
|-----
*/
void Host_Adapter_Inquiry( BYTE Host_Adapter_Num )
{
    /*--- Host Adapter Inquiry ---
    Clear_SRB();
    srb.Command = 0;
    srb.Host_Adapter_Num = Host_Adapter_Num;
    srb.Request_Flag = 0; // Direction Bits is 00
    Call_ASPI();
    if( srb.Status != 1 ) { // status=1:complete / status=0x82:error
        dsp_ASPI_status();
    }
    memcpy( &srb_ha_inquiry, &srb.Target_ID, sizeof(SRB_HA) );

    /*--- Display information [for debug] ---
    #ifdef DEBUG SCSI
        printf( "YnASPI HOST ADAPTER INQUIRY COMMANDYn" );
        printf( " Number of Host Adapters : %dYn", srb_ha_inquiry.
        printf( " Target ID of Host Adapter : %dYn", srb_ha_inquiry.
        printf( " SCSI Manager ID : %16sYn", srb_ha_inquiry.
        printf( " Host Adapter ID : %16sYn", srb_ha_inquiry.
        // printf( " Host Adapter Unique Parameters : %16sYn", srb_ha_inquiry.
    #endif
}

/*title
-----
| モジュール名 : Get_Device_Type [ASPI Command Code = 1]
| 機 能 : SCSI接続装置のタイプ (1バイト)を得る。
| コーディング・タプル : WORD Get_Device_Type(
|          BYTE Host_Adapter_Num,
|          BYTE Scsi_ID,
|          BYTE Scsi_LUN )
| 引 数 : BYTE Host_Adapter_Num = ホストアダプタID
|          BYTE Scsi_ID = SCSI接続装置のSCSI ID
|          BYTE Scsi_LUN = SCSI接続装置のLUN
| 関 数 値 : 0 = 正常終了 / -1 = SCSI接続装置がインストールされていない
| 構 造 体 : struct srb
| インクルードファイル : stdio.h
| 作 成 者 : 1993 Japan Novel Corp.
| 備 考 : このASPIコマンドは必ず0以外のステータスを返す。
|-----

```

```

*/
WORD Get_Device_Type(
    BYTE Host_Adapter_Num,
    BYTE Scsi_ID,
    BYTE Scsi_LUN )
{
    /*--- Get Device Type ---
    Clear_SRB();
    srb.Command = 1;
    srb.Host_Adapter_Num = Host_Adapter_Num;
    srb.Request_Flag = 0; // Direction Bits is 00
    srb.Target_ID = Scsi_ID;
    srb.Target_LUN = Scsi_LUN;
    Call_ASPI();
    if( srb.Status != 1 ) { // status=1:complete / status=0x82:error
        dsp_ASPI_status();
        return -1;
    }

    /*--- Display information [for debug] ---
    #ifdef DEBUG SCSI
        printf( "YnASPI GET DEVICE TYPE COMMANDYn" );
        printf( " Peripheral Device Type : %dYn", (int)((char *)&srb+10) );
    #endif

    return 0;
}

/*title
-----
| モジュール名 : Scsi_ID_Request [ASPI Command Code = 2]
| 機 能 : SCSI接続装置とのデータ転送を行う。
| コーディング・タプル : WORD Scsi_ID_Request(
|          BYTE Host_Adapter_Num,
|          BYTE Req_Flag,
|          BYTE Scsi_ID, BYTE Scsi_LUN,
|          BYTE *Data_Bp, long Data_Len,
|          BYTE Sense_Len,
|          BYTE *Cdb, BYTE Cdb_Len )
| 引 数 : BYTE Host_Adapter_Num = ホストアダプタID
|          BYTE Req_Flag = SCSIリクエストフラグ (DVK Page 2-8参照)
|          Direction Bits = 11 -> Req_Flag = 0x10
|          Direction Bits = 10 -> Req_Flag = 0x10
|          Direction Bits = 01 -> Req_Flag = 0x00
|          BYTE Scsi_ID = SCSI接続装置のSCSI ID
|          BYTE Scsi_LUN = SCSI接続装置のLUN
|          BYTE *Data_Bp = SCSI接続装置とのデータ転送エリアへのポインタ
|          long Data_Len = SCSI接続装置との転送データの長さ
|          BYTE Sense_Len = センサデータのSRBの長さ
|          BYTE *Cdb = SCSI CDB (コマンド記述ブロック)へのポインタ
|          BYTE Cdb_Len = SCSI CDB (コマンド記述ブロック)の長さ
| 関 数 値 : SCSIコマンドステータス (ホストアダプタステータス(MSB)・データ・イン・ステータス(LSB))
| 構 造 体 : struct srb
| インクルードファイル : string.h
| 作 成 者 : 1993 Japan Novel Corp.
| 備 考 :
|-----
*/
WORD Scsi_ID_Request(
    BYTE Host_Adapter_Num,
    BYTE Req_Flag,
    BYTE Scsi_ID, BYTE Scsi_LUN,
    BYTE *Data_Bp, long Data_Len,
    BYTE Sense_Len,
    BYTE *Cdb, BYTE Cdb_Len )
{
    WORD check_status = 0;

    Clear_SRB();
    srb.Command = 2;
    srb.Host_Adapter_Num = Host_Adapter_Num;
    srb.Request_Flag = Req_Flag;
    srb.Target_ID = Scsi_ID;
    srb.Target_LUN = Scsi_LUN;
    srb.Data_Len = Data_Len;
    srb.Sense_Len = Sense_Len;
    srb.Data_Bp = Data_Bp;
    srb.Cdb_Len = Cdb_Len;
    memcpy( &srb.Cdb, Cdb, srb.Cdb_Len );
    Call_ASPI();
    check_status = ((int)srb.Host_Adapter_Sta << 8) | srb.Target_Status;
    return check_status;
}

```


2.

```

WORD      Reset_SCSI_Device(
        BYTE Host_Adapter_Num,
        BYTE Scsi_ID,
        BYTE Scsi_LUN )
{
    Clear_SRB() ;
    srb.Command          = 4 ;
    srb.Host_Adapter_Num = Host_Adapter_Num ;
    srb.Request_Flag     = 0x18 ;           // Direction Bits is 11
    srb.Target_ID        = Scsi_ID ;
    srb.Target_LUN       = Scsi_LUN ;
    Call_ASPI() ;
    if( srb.Status != 1 ) {
        dsp_ASPI_status() ;
        return -1 ;
    }
    return 0 ;
}

```

1

```

        Scsi_ID,           // BYTE Scsi_ID
        Scsi_LUN,         // BYTE Scsi_LUN
        data_bp,          // BYTE *Data_Bp
        (long)data_len,   // long Data_Len
        SENSE_LEN,        // BYTE Sense_Len
        cdb_inquiry,      // BYTE *Cdb
        6 );              // BYTE Cdb_Len
    }

void main( int argc, char **argv )
{
    BYTE    Host_Adapter_Num = 0;
    BYTE    Scsi_ID = 0;
    BYTE    Scsi_LUN = 0;
    WORD    er ;
    char    buf[36], a[16], b[16];

    if ( argc == 1 ) {
        printf( "Ynusage : aspical (SCSI ID of device)\n" );
        exit( 1 );
    } else if ( *argv[1] < '0' || *argv[1] > '7' ) {
        printf( "YnSCSI ID is 0 - 7\n" );
        exit( 2 );
    } else {
        Scsi_ID = atoi( argv[1] );
    }

    /*--- Initialize ASPI ---
    if( Get_ASPI_Entry() ) {
        exit(
    }

    /*--- Inquiry for host adapter ---
    Host_Adapter_Inquiry( Host_Adapter_Num );
    Get_Device_Type( Host_Adapter_Num,
                    Scsi_ID,
                    Scsi_LUN );

    /*--- Test Unit Ready Command for SCSI device ---
    if( ( er = Test_Unit_Ready( Host_Adapter_Num,
                               Scsi_ID,
                               Scsi_LUN ) ) ) {
        SCSI_err( er );
    } else {
        printf( "YnASPI SCSI I/O COMMAND (Test Unit Ready Command)\n" );
        printf( "    Status (HEX)                : %x\n", er );
    }

    /*--- Inquiry Command for SCSI device ---
    if( ( er = Inquiry( Host_Adapter_Num,
                       Scsi_ID,
                       Scsi_LUN,
                       buf,
                       sizeof( buf ) ) ) ) {
        SCSI_err( er );
    } else {
        printf( "YnASPI SCSI I/O COMMAND (Inquiry Command)\n" );
        itoa( (int)( buf[0] ), a, 16 );
        itoa( (int)( buf[2] ), b, 16 );
        printf( "    Status (HEX)                : %x\n", er );
        printf( "    Qualifier / Device Type (HEX)   : %s\n", a );
        printf( "    ISO / ECMA / ANSI Version (HEX)  : %s\n", b );
        printf( "    Inquiry Data                : %24s\n", &buf[8] );
    }
}

```


4.3

WS/UNIX上でSCSIドライバを作成する

高須 俊介

まず一般的なUNIXのSCSIデバイス・ドライバの構造と動作について手短かに解説したあと、SunOS 4.1を実例にSCSAにもとづいたターゲット・ドライバの作成法を紹介する。SCSAは、SunOSで定義されているSCSI用のドライバ構造で、これを利用すると、ターゲット・ドライバをつくるだけで新しいデバイスを接続することができる。後半でSolarisやHP-UXでの対処法、その他のQ&Aにもふれている。
(編集部)

ハードウェアにアクセスする何らかのプログラムを作成しようとした場合には、もちろんプログラム上でそのハードウェアにアクセスするための方法を知らなければなりません。その方法を提供してくれるのは使用しているOSかもしれませんし、あるいはその方法自体を作成しなくては行けないかもしれません。いずれにしてもこのような方法を実現しているソフトウェアのことを広い意味でそのハードウェア(ターゲット・デバイス)に対するデバイス・ドライバと呼びます。

本稿では代表的なUNIXワークステーションである、サン・マイクロシステムズ社のSPARCstation上でのSCSIターゲット・デバイス用のデバイス・ドライバの作成例について述べます。なお、本稿の内容はSPARCstation2上で動作するSunOS 4.1.3を前提としています。

1

UNIXにおけるデバイス・ドライバ

最初にUNIX OSにおける一般的なデバイス・ドライバについて説明します(図17)。ここでの内容はSunOSにかぎらず多くのUNIX OSに共通です。

UNIXにはデバイスは存在しません! すべてはファイルとして扱われます! MS-DOSなどのUNIX以外のOSを利用しているプログラマはこのような言い方をされると驚いてしまうでしょう。実際、MS-DOSのA:, B:, ...のようにデバイスに特定の表現は、UNIXでは見つかりません。でも、安心してください。これは表面上のことで、デバイスは決められたディレクトリ/devの下にある特殊なファイルとしてアクセスすることができます。UNIXでは、ユーザのプロセス(UNIXではアプリケーション・プログラムのことをプロセスと呼ぶ)がデバイスを通常のファイルとまったく同じ形式でオープン/クローズできるものとして扱えるようにするため、このような名前付けをしているのです。

図18はディレクトリの内容を表示するためのUNIXコマンドls -lを使用して/devディレクトリの中身を見たものです。一般のファイルと同様にリストがとれることがわかります。しかし実体が異なるものをどうやって判別しているのでしょうか。じつはOSのカーネル(OSの中心をなすプログラムでメモリ内につねに存在しているものと考えてください)がこの作業を行ってくれます。カーネルはユーザのプロセ

図17 UNIXにおけるデバイス・ドライバの位置づけ



スにいろいろなサービスを提供してくれるプログラムです。

ユーザのプロセスがOSの提供する機能(サービス)を利用する場合にはシステム・コールと呼ばれるC言語の関数を利用します。プロセスがOSの提供するサービスであるファイルに対する操作をシステム・コールを使用して要求すると、カーネルはそのファイルの本当の姿を識別し、それがデバイス特殊ファイルであるならば、対応するドライバへの呼び出しに変換してくれるのです。それ以降の実際のデバイスへの操作はすべてドライバが行います。逆にデバイスはすべてドライバからの命令でのみ動作することになります。

■ ブロック型とキャラクタ型

ところで、これまですべていっしょくたにデバイスとってききましたが、UNIXでデバイスとして扱えるものにはまったく性質の違うものがいろいろとあります。たとえばA-D変換器やマウスのように入力専用の装置、プリンタやプロッタのような出力専用装置、ディスクやCD-ROMのようにその中にファイル・システム(ディレクトリ構造をもつもの)が作成できる装置、あるいはアレイ・プロセッサやグラフィック・アクセラレータなどの装置です。じつはRAMディスクのように関連するハードウェアが存在しないようなデバイスを扱うことも考えられます。

図18 UNIXのlsコマンドによる/devディレクトリの内容の表示例(一部の抜粋)

maple/shun % ls -l /dev									
.....									
-rwxr-xr-x	1 root	12739	Jul 24	1992	MAKEDEV*				
crw-rw-rw-	1 root	69,	0 Dec 14	1992	audio				オーディオ装置
crw-rw-rw-	1 root	69,	1 Dec 14	1992	audioctl				
.....									
crw-rw-rw-	1 root	18,	4 Nov 15	16:59	nrst0				SCSI テープ (巻き戻しなし)
crw-rw-rw-	1 root	3,	2 Dec 17	15:29	null				ヌル・デバイス
.....									
crw-r-----	1 root	17,	0 Dec 14	1992	rsd0a				SCSI ディスク 0 の 各パーティション (キャラクタ型)
crw-r-----	1 root	17,	1 Dec 14	1992	rsd0b				
crw-r-----	1 root	17,	2 Dec 14	1992	rsd0c				
crw-r-----	1 root	17,	3 Dec 14	1992	rsd0d				
.....									
crw-r-----	1 root	17,	8 Dec 14	1992	rsd1a				SCSI ディスク 1 の 各パーティション (キャラクタ型)
crw-r-----	1 root	17,	9 Dec 14	1992	rsd1b				
crw-r-----	1 root	17,	10 Dec 14	1992	rsd1c				
crw-r-----	1 root	17,	11 Dec 14	1992	rsd1d				
.....									
cr-xr-xr-x	1 root	58,	0 Dec 14	1992	rsr0*				SCSI CD-ROM (キャラクタ型)
cr-xr-xr-x	1 root	58,	8 Dec 14	1992	rsr1*				
.....									
crw-rw-rw-	1 root	18,	0 Nov 18	15:59	rst0				SCSI テープ (巻き戻しあり)
crw-rw-rw-	1 root	18,	1 Dec 14	1992	rst1				
.....									
brw-r-----	1 root	7,	0 Dec 14	1992	sd0a				SCSI ディスク 0 の 各パーティション (ブロック型)
brw-r-----	1 root	7,	1 Dec 14	1992	sd0b				
brw-r-----	1 root	7,	2 Dec 14	1992	sd0c				
brw-r-----	1 root	7,	3 Dec 14	1992	sd0d				
.....									
br-xr-xr-x	1 root	18,	0 Dec 14	1992	sr0*				SCSI CD-ROM (ブロック型)
br-xr-xr-x	1 root	18,	8 Dec 14	1992	sr1*				
.....									
crw-----	2 root	3,	5 Dec 14	1992	vmel6				VME バス
crw-----	2 root	3,	5 Dec 14	1992	vmel6d16				

↑
デバイスの型

↑
メジャー番号

↑
マイナー番号

UNIX ではこれらのデバイスを、

▷ブロック型デバイス

▷キャラクタ型デバイス

に分けています。

ブロック型デバイスというのは別名、構造型デバイスとも呼ばれ、デバイスが保持するデータが構造を持ち、その構造にしたがってデータをランダムにアクセスすることが要求されるもの、簡単にいえばディスクのようにその中にファイル・システムを構築できるものです。

一方、**キャラクタ型デバイス**とは非構造型デバイスとも呼ばれ、通常はストリーム・テープ装置のようにシーケンシャルにデータを入出力するための装置をいいます。図18で型として示された文字(b=ブロック型、c=キャラクタ型、など)がこの分類を示しています。

■ デバイス・スイッチ

カーネルは現在利用できるデバイス・ドライバの一覧表をこの分類ごとにデバイス・スイッチと呼ばれる構造体の配列(ブロック・デバイス用が bdevsw, キャラクタ・デバイス用が cdevsw)の形で保持していて、指定されたデバイス特殊ファイルとそれに対応するデバイス・ドライバを対応づけています。図2のメジャー番号がデバイス・スイッチの配列に対するインデックスとして使われ、特定のデバイス・ドライバを選択します。マイナー番号はデバイス・ドライバにパラメータとして渡され、その使われ方はデバイス・ドライバに依存して変わります。

デバイス・スイッチの構造体の中身は、じつはデバイス・ドライバが提供する関数へのポインタです。カーネルはこのポインタを使用してドライバの提供する機能(デバイスの open, close, read, write, ioctl など)を呼び出すのです。

2

SunOSにおける SCSIデバイス・ドライバ

さてここで話を SCSI に移しましょう。ハードウェアへアクセスする手段を実現するという意味でデバイス・ドライバをとらえた場合、SCSI インターフェースで接続された何らかのターゲット・デバイス(テープと

かディスクなど)のドライバが扱うべきなのは、SCSI インターフェースの制御についてでしょうか?それともターゲット・デバイスの制御でしょうか?

答えはもちろんその両方です。SCSI インターフェースの制御を行うだけのドライバではユーザのプログラムはターゲット・デバイスに特有の機能をすべて自分で解決し、さらにそれを SCSI 独特のコマンドに変換する作業も行わなくてはなりません。これではドライバの機能が十分ではありません。一方でターゲット・デバイスの制御を行うドライバは、当然、内部で SCSI インターフェース・ハードウェアを制御しなければなりません。それがターゲットごとに異なるドライバで行われているならば、ドライバ間での SCSI インターフェースの取り合いを管理するのは大変です。

■ SCOSA=SCSI ターゲット・ドライバ+ホスト・アダプタ・ドライバ

これらの問題を解決するにはどうやら SCSI 用に適したドライバの構造を作り上げる必要があります。SunOS ではこのような構造として SCOSA (Sun Common SCSI Architecture) を定義しています。SCOSA で定義されている構造を図19に示します。

SCOSA では二つの主要なソフトウェア部品とその間のソフトウェア・インターフェースを定義しています。上位レベルのソフトウェア部品は一つ以上の **SCSI ターゲット・ドライバ**です。その中心となる機能は UNIX カーネルからの I/O リクエストを UNIX 上のアプリケーションが指定した周辺機器に適した SCSI コマンドに変換することにあります。下位レベルのソフトウェア部品は一つ以上の **ホスト・アダプタ・ドライバ**から構成されています。

ホスト・アダプタ・ドライバの機能はいろいろなローカル・リソースを管理してターゲット・ドライバからの SCSI コマンドを受け取り、特定の SCSI ターゲットへ送り、そのコマンドが必要とするすべてのデータの授受を行い、状態を検知し、ターゲット・ドライバにコマンドが完了したか失敗したかを伝えます。

ターゲット・ドライバから送られてくるのはどんな SCSI コマンドでもかまいませんし、それを予知できませんから、ホスト・ドライバはコマンド・ディスクリプタ・ブロック(CDB)の特別な内容についてはまったく知る必要はありません。ターゲット・ドライバが要求される機能に応じた正しい SCSI コマンドを生成

する責任を負うのです。ホスト・アダプタ・ドライバにとっては、SCSI の CDB は指定された SCSI ターゲットに対して、標準のプロトコルにしたがって SCSI バスを通じて送るべき、たんなるバイトの列と見えるだけです。

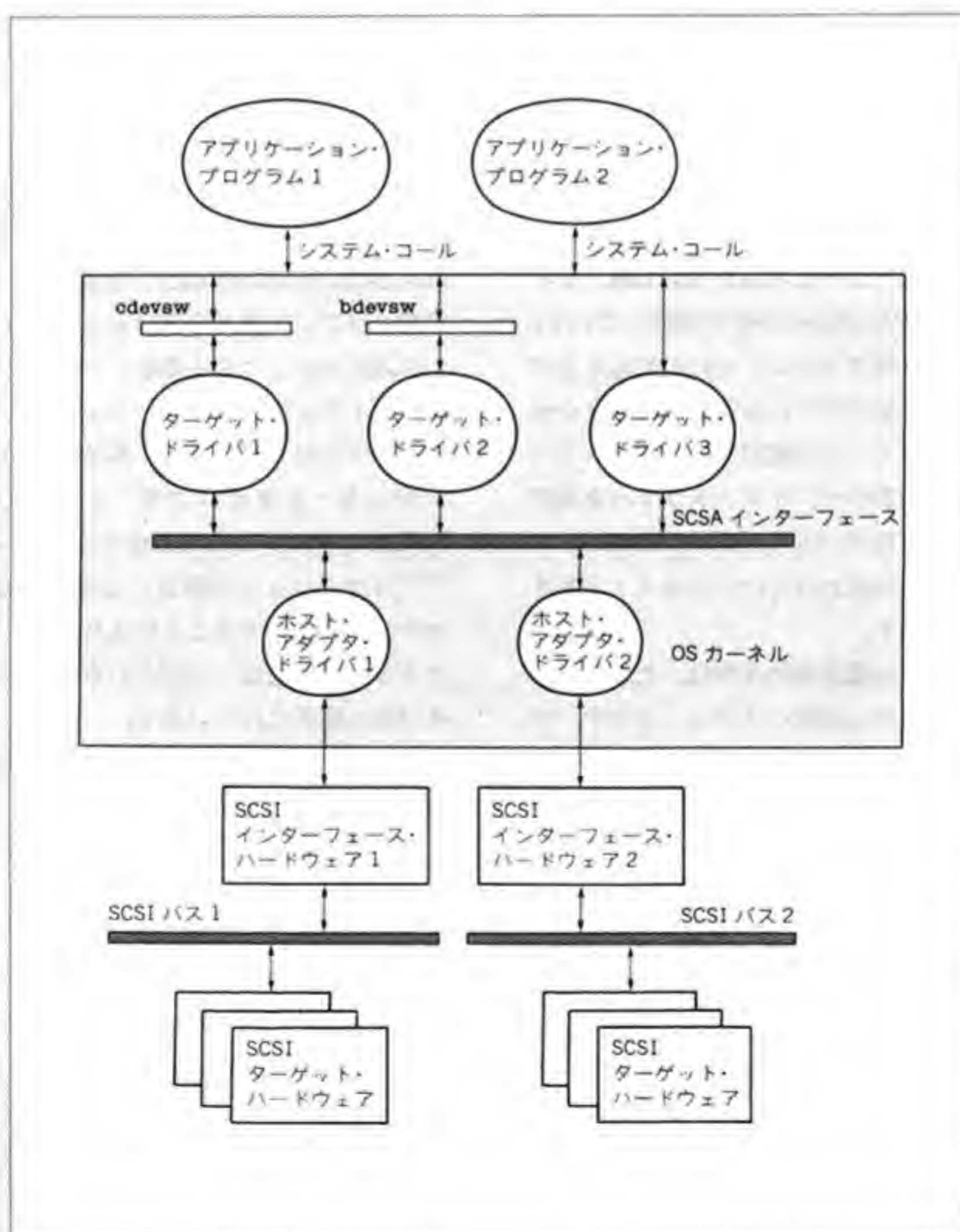
このような構造を採用することによって、SCSI ホスト・アダプタ・ドライバとホスト・アダプタ・ハードウェア自体の実装方法に依存しないターゲット・ドライバを作成したり、ターゲット・ドライバからのコマンド・シーケンスの選び方や、アプリケーションに依存しないすべての種類のターゲットをサポートする

ホスト・アダプタ・ドライバを作成することを可能にしています。

■ プログラマはターゲット・ドライバを作るだけ

簡単にいうと SCSI インターフェースのハードウェアに依存し、対応する SCSI バスの管理を行う部分と、実際にその SCSI バスに接続するデバイスに依存した部分を別々のドライバで担当する構造です。ワークステーションに組み込みの SCSI インターフェース・ハードウェア用のホスト・アダプタ・ドライバは OS の一部として供給されているので、プログラマは、通常は

図19 SCOA(Sun Common SCSI Architecture)の構造



ターゲット・ドライバを作成すればよいのです。

じつは、このことはプログラマの作業を大幅に軽減してくれます。というのはUNIXでは仮想アドレスを使っていて、カーネルは頻繁に物理アドレスと仮想アドレスのマッピングを変更しているのです。ホスト・アダプタ・ドライバのように本質的に物理アドレスを直接アクセスしなければならないドライバを作成するには、つねにこれを意識して管理しなければなりません。

3

アプリケーション・プログラムから ドライバへのインターフェース

先に説明したようにアプリケーション・プログラムからデバイスへアクセスするにはカーネルが供給するシステム・コールを使用します(図17)。そのおもなところはopen(), close(), read(), write(), ioctl()などのC言語のライブラリ関数です。

ここで、これらの関数が特定のデバイスやSCSIインターフェースを使用していることをとくに意識させないことに注意してください。そのようなデバイスに特有の部分はすべてドライバが吸収するのがUNIXのやり方なのです。通常のファイルに対するオープン/クローズ、読み/書きと同様の動作をするようにデバイス・ドライバが適当なSCSIコマンドに置き換え、SCSIプロトコルを管理し、実際のデータ転送を行わなくてはなりません。特定のデバイスに対して実際にデータを読んだり、書いたりした場合に何が起るのかはデバイス・ドライバをどう書くかによって決まるのです。したがってデバイス・ドライバの設計はそのような意味付けを決めることがもっとも重要になります。とはいってもデータの読み/書きでは意味付けができないような動作をデバイスに行わせたい場合もあります。たとえば、テープ装置に対してテープの巻き戻しを実行させることをデータの読み/書きに対応させるのはいかにも不自然です。このような場合に使われるのがioctl()関数です。ioctl()関数で行えることは、したがって本質的に特定のデバイスに応じて変わるものであり、それはデバイス・ドライバによって実現しなければなりません。

4

ターゲット・ドライバの例

それでは実際の例を使ってターゲット・ドライバの説明を続けましょう。この例では高速で多チャネルのロジック・アナライザ・ハードウェアをSCSIインターフェースを使用してSunワークステーションに接続します。ドライバに対する要求は単純で、たんにアプリケーション・プログラム(実際のこのプログラムはロジック・アナライザのヒューマン・インターフェースと収集されたロジック・データの解析や逆アセンブルを行う大きなプロセス)からのデータをデバイスに送り、デバイスからのデータを読み取り、アプリケーションに返すことができればよいといったものです。ただし、1回に転送するデータの量は最大48Kバイト程度の量となります。また、データの送受信においてはデータの内容に応じて異なったパラメータによるSCSIコマンドを実行しなければならないため、read(), write()関数ではなく、ioctl()関数を使用します。

残念ながら、ここにドライバすべてのソース・コードを提示し、詳細な説明を加えることは、ほぼ書籍1冊分のスペースを必要とするうえ、OSに対する高度な知識を必要としてしまうため断念せざるを得ません。かわりにドライバ内に記述しなければならない主要な関数のソース・コードを提示するにとどめます(p. 111~のリスト3)。なお、より詳細な説明とサンプルのソース・コードは、サン・マイクロシステムズ社のコンサルティング・プロダクトとして購入可能です(SCSIドライバ・スタータ・キット パーツ番号: CONSULT-SIP 連絡先: 日本サン・マイクロシステムズ株式会社・サービス本部 ビジネス・サポート部 ☎ 03-3221-2453)

ドライバがサポートしなければならない関数はつぎのとおりです。

```
int xxslave(devp)
struct scsi_device *devp;
```

システム立ち上げ時に呼び出されます。この関数が行わなければならないことはターゲット・デバイスが存在することを確認し、それに対して必要な初期化を行うことです。初期化に成功したときには“1”を、そ

れ以外のときには“0”を返します。

```
int xxattach(devp)
struct scsi_device *devp;
```

xxslave が成功した場合、それに続いて呼び出されます。通常はターゲット・デバイスからそれに特有の情報を得るために使われます。

```
int xxopen(dev, flag)
dev_t dev;
int flag;
```

デバイスに対応するファイルをオープンするたびに呼び出されます。指定されたデバイスが存在することを確認し、必要であればそのデバイスへのアクセスのための初期化を行います。

```
int xxclose(dev)
dev_t dev;
```

デバイスに対応するファイルをクローズするたびに呼び出されます。指定されたデバイスが存在することを確認し、必要であればそのデバイスへのアクセスを終了するための処理を行います。

```
int xxread(dev, uio)
dev_t dev;
struct uio *uio;
```

データをデバイスから読み取るために呼び出されます。

```
int xxwrite(dev, uio)
dev_t dev;
struct uio *uio;
```

データをデバイスに書き込むために呼び出されます。

```
xxstrategy(bp)
register struct buf *bp;
```

データの転送時に OS が提供する buf 構造体を使用する場合、physio() 関数から呼び出されます。この関数はもともとブロック型デバイス用に使われていたもので、入出力要求を適当に分割し、デバイスに対して最適な順番で実際の転送が行われるようにします。たとえば、ディスクのようなランダム・アクセス・デバイスではデータの読み/書きの順序を実際にデータが格納されているセクタの位置によって変えてやることで早くできることがあります。この例ではキャラクタ型デバイスを想定していますので、このような本来の機能は必要としませんが、割り込みによるデータの転送を管理します。デバイスがデータ転送可能かどうかを調べ、データ転送を開始し、データ転送が完了し

割り込みルーチンがシグナルを出すのを待ちます。

```
int xxioctl(dev, cmd, data, flag)
dev_t dev;
int cmd;
caddr_t data;
int flag;
```

デバイスに特有の I/O を制御するために呼び出されます。一般的な read(), write() の意味に置き換えられないデバイス制御はこの関数で実行することができます。

```
static void xxstart(devp)
register struct scsi_device *devp;
```

デバイスへのデータの送受信を開始する関数です。デバイス・ドライバ内部からのみ使われます。

■ SCSI を使用したコマンドの実行シーケンス

プログラムがシステム・コールを使用してカーネルに対して I/O 機能を要求するとカーネルはそれをターゲット・ドライバへの要求に置き換えます。ターゲット・ドライバは SCSI ターゲット・デバイスが受け取ったときに要求された機能が実行されるように SCSI コマンドを準備します。

つぎにターゲット・ドライバは必要になるメモリのブロックを確保するための関数を実行し、そのブロックの空白にホスト・アダプタ・ドライバへのコマンドを記述します。さらにターゲット・ドライバはホスト・ドライバが用意している関数を呼び出して、コマンドを実行させるために必要なその他のリソースを得ます。

実際にはホスト・アダプタ・ドライバがそのリソースを管理しているため、それをただちに確保するか、または後のいつかの時点で確保するかは、ホスト・アダプタ・ドライバが決めることになります。最後に、ターゲット・ドライバは記述した情報をトランスポート関数を使用してホスト・アダプタ・ドライバに渡します。いったんこの情報を渡した後は、ターゲット・ドライバは同じターゲットに対するほかの SCSI コマンドや他の接続機器へのコマンドに対応できるようにただちに開放されます。

ホスト・アダプタ・ドライバは、トランスポート関数によって必要な情報が受け取られると、そこに記述されている SCSI コマンドをもっとも効率よく実行することに専念します。ホスト・アダプタ・ドライバはすべての DMA リソース、ホスト・アダプタのハード

ウェアそして SCSI プロトコルを管理し、そのコマンドを実行します。その特定のデバイスがディスクコネク
トされている間は、ホスト・アダプタ・ドライバはそ
れ以外の SCSI デバイスへの操作を実行することがで
きますし、タグ付けされたキュー・プロトコルを使用
して SCSI デバイスへの操作をキューに置くこともで
きます。

コマンドが成功したり失敗して終了すると、ホス
ト・アダプタ・ドライバはその状態の情報を先に確保
されているメモリ領域のなかに埋め込み、ターゲッ
ト・ドライバが指定したコンプリーション・ルーチン
を呼び出します。このコンプリーション・ルーチンが、
ターゲット・ドライバにその SCSI 操作が終了したこ
とを知らせます。この時点でホスト・アダプタ・ドラ
イバは特定のコマンドに対する責任をもたなくなり、
かわりにターゲット・ドライバがその責任を引き継ぎ
ます。

ターゲット・ドライバは SCSI 操作が成功したかど
うかを示す返り値を解釈した後に、ホスト・アダプタ・
ドライバに DMA リソースと確保されていたメモリ
を開放するように要求し、それらが自分自身やその他
のターゲット・ドライバで再度利用できるようにしま
す。ターゲット・ドライバはカーネルを介して最初に
このトランザクションの要求を出したプログラムにそ
のトランザクションが終了したことを通知します。

以上のような簡単な構造によって、プロセス中のい
ろいろな時点でオーバーラップしたり、キューされたり
するたくさんの SCSI I/O の実行を可能にしています。
またこの構造によって、複雑な仮想アドレスの管理は
ホスト・アダプタ・ドライバの中に実装されたカーネ
ル・コードが行うようにして、ターゲット・ドライバ

の作成を容易にしています。さらにホスト・アダプタ・
ドライバとターゲット・ドライバ間のソフトウェア・
インターフェースが定義されたことによって、異なる
種類の SCSI インターフェース・ハードウェア、たとえ
ばコプロセッサを実装したインテリジェント SCSI イ
ンターフェース・ハードウェアを使用するホスト・ア
ダプタ・ドライバに対しても共通のターゲット・ドラ
イバの機能が実行できるようにしています。

■ 機能インターフェースのまとめ

ターゲット・ドライバ層とホスト・アダプタ・ドラ
イバ層の主要なインターフェースは関数ライブラリに
なっています。サポートされる関数は、リソースの確
保に関するもの、コマンドの転送に関するものと、転
送の情報と制御に関するものに分けられます。定義済
みの関数は表10 にまとめられます。

SCSI ターゲットにコマンドを送るためには、ター
ゲット・ドライバはホスト・アダプタ・ドライバにコ
マンドを送る前にローカルなリソースを確保しなけれ
ばなりません。これらのリソースはターゲット・ドラ
イバより下の層でホスト・アダプタ・ドライバによっ
て管理されています。実際にその機能がホスト・ドラ
イバ自体または一般的なライブラリ関数によって実行
されるかはターゲット・ドライバからは見えず、関与
できません。必要とされるリソースは構造体 `scsi_pkt`
(これには SCSI ターゲットに送られるコマンドが記
述されている)と、必要な DMA リソース(たとえば
DVMA 領域のリザーブされたトランスレーション)
です。

ホスト・アダプタ・ドライバは要求に応じて DMA
リソースを管理し、実際に必要になるまでそれを確保

表10 定義済みのインターフェース関数

<code>scsi_realloc()</code> <code>scsi_pktalloc()</code> <code>scsi_dmaget()</code> <code>scsi_dmafree()</code>	リソースの確保
<code>pkt_transport()</code>	コマンドの転送
<code>scsi_abort()</code> <code>scsi_reset()</code> <code>scsi_ifgetcap()</code> <code>scsi_ifsetcap()</code>	転送の情報と制御

表11 構造体 `scsi_pkt` と DMA リソースを
得るための関数

<code>scsi_poll()</code>	ポーリングによるコマンドの実行
<code>get_pktiopb()</code> <code>free_pktiopb()</code>	
<code>scsi_slave()</code>	スレーブ・ルーチンの実行
<code>makecom_g0()</code> <code>makecom_g1()</code> <code>makecom_g0_s()</code>	グループ0用 <code>scsi_pkt</code> の準備 グループ1用 <code>scsi_pkt</code> の準備 テープ用グループ0の準備

しないかもしれません。そこで表11に示すように、これらの二つのリソースを得るためのいくつかの関数が用意されています。

■ 構造体について

表11に示した関数は二つの重要な構造体を使用しています。構造体 `scsi_address` は要求された関数の中で、どのホスト・アダプタとデバイスを使用しているかを示します。構造体 `scsi_pkt` は SCSI コマンドを実行するうえで管理し実行し、状態を返すために必要な情報を与えます。

■ ドライバの組み込み

UNIX のシステムに SCSI ターゲット・ドライバを組み込む方法は一般的につきのようなステップにしたがいます。

- ・システム用のディレクトリにドライバ・ソース・ファイルをコピーする。
- ・システムのコンフィギュレーション・ファイルにド

ライバを追加する。

ここでは、その詳しい方法にはふれません。詳細については、サンのマニュアルを参照してください。ここで注意したいことは、一般に UNIX ではドライバの組み込みのためにドライバのソース・コードが必要なことです。システムのコンフィギュレーション・ファイルと呼ばれる C のソース・ファイルを修正してドライバを組み込むことを宣言する変数や定義を追加したあとに、ソース・コードをコンパイルし、カーネルにリンクする作業が必要になるのです。これは、パソコンの場合よりもユーザにはるかに高度な知識と注意を要求するため、UNIX を気軽に使用できない原因の一つと考えられます。

5

その他のワークステーションの場合

本稿では、現在もっともユーザが多いと思われる



UNIX/SCSI に関する Q&A

以下は筆者が良く受ける UNIX 上での SCSI に対する質問とそれに対する回答です。

Q UNIX での標準 SCSI プロトコル・レベルは？

A とくに標準はありません。ベンダやドライバ自体に依存しています。

Q コマンド・キューイングを使ってマルチプロセスでのデバイス共有効率を上げている事例はあるか？

A 行われています。とくにディスク・ドライバで重要な部分になります。

Q UNIX がサポートしていない SCSI デバイスを接続するとどうなるか？

A 他のデバイスとアドレスがぶつからないかぎりなにも起こらないでしょう。多くの UNIX では立ち上げ時にデバイスの存在を Inquiry 命令によって確認しています。したがって、期待した返答を返さないデバイスは存在しないものとみなされるでしょう。

Q システムが稼働中にターゲット・デバイスの電

源をいれて立ち上げたら、このデバイスは使用できるか？

A 先の質問の答えのとおり、通常は立ち上げ時のみデバイスの存在の確認を行うため、特殊なドライバでないかぎり使用できないでしょう (HP-UX の `scsi_ctl` ドライバをユーザのプログラムから使用する場合には可能)。

Q UNIX のホスト・マシンがターゲットになることはあるか？なりうるのか？

A 私の知るかぎりありません。通常は、UNIX マシンが SCSI ホストとなることを前提に環境が作られているため、できるとしてもかなりの労力が必要となるでしょう。

Q 他のコンピュータ OS (Macintosh など) 用のディスクは UNIX と共有できるか？

A これはすべてドライバ次第です。サン・マイクロシステムズ社ではコンサルティング・プロダクトの一つとして IBM PC 互換のフロッピー・ディスクをサポートする SCSI ドライバを販売しています。

Sun の SunOS 4.1 上での SCSI ドライバについて説明しました。サンの新しい OS である Solaris 2.x においても本稿の内容の大筋は変わりません(ドライバの組み込みの際、ソース・コードを必要としなくなるなどの改善点はある)。

もう一つ広く使われている UNIX WS である HP 9000・700 シリーズでは最新の OS である HP-UX 9.0 より scsi_ctl ドライバがサポートされるようになりました。このドライバはターゲットとなるデバイスを特定しません。したがって read(), write() のような特定のターゲット・デバイスによって異なる動作をしなければならないシステム・コールを介して自由に SCSI コマンドの送受ができる環境を提供してくれます。このなかには同期/非同期でのデータ転送も含まれています。これを使うことによって標準でサポートされない SCSI デバイスをワークステーションに接続し、それをコントロールするソフトウェアを比較的簡単に書くことができるようになっています。ただし、その場合には、ユーザのプログラムが特定のデバイスに依存した制御を行わなくてはなりません。

■ まとめ

ディスク、テープ・ドライブ(1/4 インチ・テープ、8 mm, DAT), CD-ROM など標準的なデバイスについ

ては、そのいろいろな機種についてのドライバがすでに OS に組み込まれた形で提供されているため、多くの場合、ユーザがあえてプログラミングする必要はないかもしれません。また、UNIX のデバイス・ドライバをプログラミングすることはなかなか難しいのが現状です。これは、UNIX がマルチユーザ・マルチタスク OS であることと、標準となるデバイス・インターフェースがなかったことによることが大きいと考えます。しかしながら SCSI という良く定義されたインターフェース・プロトコルが実際上 UNIX ワークステーションでの標準となってきたため、いくらかでもプログラムをすることが楽になってきているのではないかと思います。本稿がそのようなプログラミングをする上での一助になれば幸いです。

参考文献

- 1) *SCSA SUN common SCSA Architecture*, P/N : 800-4701-10, Sun Microsystems
- 2) *Implementation Guide SUN Common SCSA Architecture*, P/N : 800-4700-10, Sun Microsystems
- 3) *HP-UX Release 9.0 Reference Manual*, Hewlett-Packard Co.
- 4) *Writing Device Driver* (SunOS 4.1 マニュアル), Sun Microsystems

たかす・しゅんすけ ㈱東陽テクニカ 技術本部

リスト 3 ドライバ内に記述しなければならない主要な関数のソース・コード ①

```
#include "cls.h" /* このドライバ例ではドライバ名を cls としている */
#if NCLS > 0 /* 対応する関数も clsopen() 用に頭の xx を cls に置き換えられている */
/* 引き数のチェックなどの詳細はほとんど省略し、スケルトンのみ */

#include <scsi/scsi.h> /* SCSA が使用するヘッダ・ファイル */
#include <scsi/targets/clsdef.h> /* ドライバに固有の定義はこのファイルに入れる */
#include <vm/hat.h>
#include <vm/seg.h>
#include <vm/as.h>

#define DNAME devp->sd_dev->devi_name
#define DUNIT devp->sd_dev->devi_unit
#define CNAME devp->sd_dev->devi_parent->devi_name
#define CUNIT devp->sd_dev->devi_parent->devi_unit
/*
 * buf 構造体のバック・ポインタが SCSI パケットで使われる。
 */
#define BP_PKT(bp) ((struct scsi_pkt *)bp->av_back)
#define SCBP(pkt) ((struct scsi_status *) (pkt)->pkt_scbp)
#define SCBP_C(pkt) ((*(pkt)->pkt_scbp) & STATUS_MASK)
#define CDBP(pkt) ((union scsi_cdb *) (pkt)->pkt_cdbp)
#define TGT(devp) (devp->sd_address.a_target)
#define LUN(devp) (devp->sd_address.a_lun)
#define CLSPDS ((struct scsi_cls *) (devp)->sd_private)

#define CLSUNIT(d) (minor((d))) /* マイナー番号のコード化が必要であればここ
                                * である */

#define newstate(s) CLSPDS->un_last_state=CLSPDS->un_state, CLSPDS->un_state=(s)
#define VIDSZ 8
#define PIDSZ 16
/*
 * SunOS では 1 回の DMA 転送で送れるサイズは 63 K まで。
 */
#define CLS_MAXREC (63 * 1024)
#define CLS_MAXUNITS 8

/*
 * このドライバに固有のデータ
 */
struct scsi_device *clsunits[CLS_MAXUNITS];

int ncls = NCLS;
static int clsprl;

/*
 * デバイス・ドライバがサポートすべき関数とその関数へのベクタを保持する構造体
 * を示す。
 * デバイス・ドライバはこれらの関数の集合体と考えられる。
 */
int clsslave(), clsattach(), clsopen(), clsclose(), clsread(), clswrite();
int clsstrategy(), clsioctl(), clsinfo();
extern int nulldev(), nodev();
struct dev_ops cls_ops = {
    1,
    clsslave,
    clsattach,
```


リスト3 ドライバ内に記述しなければならない主要な関数のソース・コード ②

```

    clsopen,
    clsclose,
    clsread,
    clswrite,
    clsstrategy,
    nodev,
    nulldev,
    clsioctl,
};

static void clsintr(), clsdone(), cls_make_cmd(), clserrmsg();
static void clean_print();

/*
 * clsslave
 * この関数はシステムの立ち上げ時にOSから自動的に呼び出され、
 * 指定されたデバイスがターゲットIDに存在することを確認する。
 * 存在が確認されると、それ以降このドライバを介してデバイスを使
 * 用することができるようになる。
 */
int
clsslave(devp)
register struct scsi_device *devp;
{
    char        vpid[VIDSZ+PIDSZ+1];
    int         status;
    struct scsi_pkt *rsp, *get_pktiopb();

    clsunits[DUNIT] = devp; /* fill in our array */
    clspri = MAX(clspri, lptospl(devp->sd_dev->devi_intr->int_pri));
    /*
     * scsi_slave() はデバイスに test unit ready と inquiry 命令を送り、
     * デバイスからの返答を読みとる。
     */
    status = scsi_slave(devp, NULL_FUNC);
    switch(status) {
    case SCSI_PROBE_EXISTS:
        /*
         * inquiry 命令が成功すると、ここで、devp->sd_inq に返答が
         * 記録されている。
         * inq_dtype, inq_vid, inq_pid などのフィールドを調べ、期待
         * したターゲット・デバイスが接続されていることを確認する。
         * inq_dtype に応じたデバイス特有のチェックを 必要ならば行う。正
         * しく接続されていない場合には0を返す。この例では常に正しいデ
         * バイスが接続されているものとして扱う。
         */
        switch(devp->sd_inq->inq_dtype) {
        case DTYPE_DIRECT:
        case DTYPE_SEQUENTIAL:
        case DTYPE_PRINTER:
        case DTYPE_PROCESSOR:
        case DTYPE_WORM:
        case DTYPE_RODIRECT:
        case DTYPE_SCANNER:
        case DTYPE_OPTICAL:
        case DTYPE_CHANGER:
        case DTYPE_NOTPRESENT:
        default:
            printf("%s%d: found %s device (tgt%d, lun%d) on %s%d\n",
                DNAME, DUNIT,
                scsi_dname((int)devp->sd_inq->inq_dtype),
                TGT(devp), LUN(devp), CNAME, CUNIT);
            bcopy((caddr_t)devp->sd_inq->inq_vid, (caddr_t)vpid,
                VIDSZ);
            bcopy((caddr_t)devp->sd_inq->inq_pid,
                (caddr_t)vpid+VIDSZ, PIDSZ);
            vpid[VIDSZ+PIDSZ] = 0;
            printf("%sVendor/Product ID = %s\n",
                devp->sd_inq->inq_vid,
                devp->sd_inq->inq_pid);
            break;
        }
        break;
    case SCSI_PROBE_NONCCS: /* inquiry 命令をサポートしていないときは
        ここで対処する */
    case SCSI_PROBE_NORESP:
    case SCSI_PROBE_NOMEM:
    case SCSI_PROBE_FAILURE:
    default:
        DPRINTF(1)("%s: cls_findslave: failed, scsi_slave returned 0x%x\n", status);
        devp->sd_present = 0; /* no unit present */
        return(0);
        break;
    }
}

/*
 * ターゲットIDに期待されたデバイスが存在していることを確認した。
 * 後は、これ以降に使用するプライベート・データ領域を確保し、
 * デバイスの初期化を行う。
 */
if (!(rsp = get_pktiopb(&devp->sd_address, (caddr_t *)&devp->sd_sense,
    CDB_GROUP0, 1, SENSE_LENGTH, R_READ, NULL_FUNC))) {
    printf("%s%d: No memory for request sense buffer\n",
        DNAME, DUNIT);
    return(0);
}
makecom_g0(rsp, devp, 0, SCMD_REQUEST_SENSE, 0, SENSE_LENGTH);
rsp->pkt_pmon = -1;
rsp->pkt_comp = clsintr;
rsp->pkt_time = DFLT_CLS_TIMEOUT;

/*
 * プライベート・データ領域を確保し、初期化する。
 */
CLSPDS = (struct scsi_cls *)kmem_zalloc(sizeof(struct scsi_cls));
if (!CLSPDS) {
    printf("%s%d: No memory for device structure\n", DNAME, DUNIT);
    free_pktiopb(rsp, (caddr_t)devp->sd_sense, SENSE_LENGTH);
    return(0);
}
CLSPDS->un_rbufp = (struct buf *)kmem_zalloc(sizeof(struct buf));
CLSPDS->un_sbufp = (struct buf *)kmem_zalloc(sizeof(struct buf));
if (!CLSPDS->un_rbufp || !CLSPDS->un_sbufp) {
    printf("%s%d: can't allocate raw/special buffer\n", DNAME, DUNIT);
    if (CLSPDS->un_rbufp) {
        (void) kmem_free((caddr_t)CLSPDS->un_rbufp, sizeof(struct buf));
    }
    free_pktiopb(rsp, (caddr_t)devp->sd_sense, SENSE_LENGTH);
    (void) kmem_free((caddr_t)CLSPDS, sizeof(struct scsi_cls));
    return(0);
}
CLSPDS->un_tmpbuf = (caddr_t)kmem_zalloc(UN_TMPBUF_SIZE);
if (!CLSPDS->un_tmpbuf) {
    printf("%s%d: can't allocate un_tmpbuf\n", DNAME, DUNIT);
    (void) kmem_free((caddr_t)CLSPDS->un_sbufp, sizeof(struct buf));
    (void) kmem_free((caddr_t)CLSPDS->un_rbufp, sizeof(struct buf));
    (void) kmem_free((caddr_t)CLSPDS, sizeof(struct scsi_cls));
    free_pktiopb(rsp, (caddr_t)devp->sd_sense, SENSE_LENGTH);
    return(0);
}
CLSPDS->un_rsp = rsp;
CLSPDS->un_sd = devp;

devp->sd_present = 1; /* ユニットが存在することを記す。 */
devp->sd_dev->devi_driver = &cls_ops;
(void) clsattach(devp); /* デバイス固有の初期化を行う。 */
return(1);

/*
 * clsattach
 * 必要に応じてデバイス固有の初期化を行う。
 */
int
clsattach(devp)
struct scsi_device *devp;
{
    /*
     * 省略
     */
    CLSPDS->un_attached = 1;
    return;
}

/*
 * デバイスのオープン。デバイスの存在を確認し、デバイス固有の初期化を行う。
 */
int
clsopen(dev, flag)
dev_t dev;
int flag;
{
    register struct scsi_device *devp;
    register int unit, s;
    if (devp->sd_present == 0) {
        DPRINTF(1)("%s: clsopen: device not present, unit=%d, devp=0x%x\n",
            unit, devp);
        return(ENODEV);
    }
    DPRINTF(2)("%s: clsopen: cls%d, devp=0x%x\n", unit, devp);
}

```



```

/*
 * デバイスが初期化されていなければそれを初期化する。
 */
if (!CLSPDS->un_attached) {
    clsattach(devp);
    if (!CLSPDS->un_attached) {
        return(ENXIO);
    }
}

/*
 * デバイスがクローズされていた場合には、必要ならばオープンして初期化
 * しなおす。
 */
if (CLSPDS->un_state == CLS_STATE_CLOSED) {
    newstate(CLS_STATE_OPENING);
    /* オープンして初期化するためのコードをここに書く */
}
newstate(CLS_STATE_OPEN);
return(0);
}

/*
 * デバイスのクローズ
 */
int
clsfclose(dev)
dev_t dev;
{
    register struct scsi_device *devp;
    int unit;

    newstate(CLS_STATE_CLOSED);
    /*
     * クローズに必要な操作を行う。テープ装置の場合はテープの巻き戻し
     * などが必要となる。
     */
    return(0);
}

/*
 * 入出力のためのエントリ関数
 */
int
clsread(dev, uio)
dev_t dev;
struct uio *uio;
{
    return(clsrw(dev, uio, B_READ));
}

int
clsfwrite(dev, uio)
dev_t dev;
struct uio *uio;
{
    return(clsrw(dev, uio, B_WRITE));
}

/*
 * 直接入出力のルーチン
 * uio の値をチェックし、実際の転送はphysio を介してclsstrategy を
 * 呼び出して行う。
 */
static int
clsrw(dev, uio, flag)
dev_t dev;
struct uio *uio;
{
    struct scsi_device *devp;
    register int unit;

    /* uio の値をチェック 省略 */
    return(physio(clsstrategy, CLSPDS->un_rbufp, dev, flag, clsinphys, uio));
}

```

```

/*
 * clsstrategy
 * デバイスへコマンドを送るためのメイン関数。buf をキューにおいた後、
 * スタート・ルーチンを実行する。スタート・ルーチン(clsstart)は、
 * SCSI コマンドを用意し、DMA 転送に必要なメモリを確保した後に
 * ホスト・アダプタ・コントローラにコマンドを送る。
 */
int
clsstrategy(bp)
register struct buf *bp;
{
    register struct scsi_device *devp = clsunits[CLSUNIT(bp->b_dev)];
    register int s;
    struct buf *ap;
    s = splr(clspri);

    /*
     * buf をキューに置く
     */
    if (CLSPDS->un_quehd) {
        CLSPDS->un_quetl->av_forw = bp;
    }
    else {
        CLSPDS->un_quehd = bp;
    }
    CLSPDS->un_quetl = bp;
    CLSPDS->un_bufcnt++;

    bp->av_forw = 0;
    bp->b_flags |= (B_DONE | B_ERROR);
    bp->b_resid = 0;

    /*
     * clsstart にバッケットの確保が必要であることを伝えるため
     * bp->av_backを0に設定する。
     */
    if (bp != CLSPDS->un_sbufp) {
        BP_PKT(bp) = 0;
    }

    /*
     * スタート・ルーチンを起動する。
     */
    if (CLSPDS->un_active == NULL)
        clsstart(devp);
    (void)splx(s);

    return(0);
}

/*
 * ioctl の実行
 */
int
clsioctl(dev, cmd, data, flag)
dev_t dev;
register int cmd;
register caddr_t data;
int flag;
{
    register struct scsi_device *devp = clsunits[CLSUNIT(dev)];
    /*
     * ここでは、コマンドの種類を調べそれに応じた処理を行う。
     */
    switch(cmd) {
        case CLSIOC_GETERRC: /* エラーの詳細を調べる */
            bcopy(&cls_ok_status, (caddr_t)data, sizeof(char));
            break;
        case CLSIOC_RESET: /* ターゲットのリセット */
            return(scsi_reset(&devp->sd_address, RESET_TARGET));
            break;
        case CLSIOC_CMD: /* 一般のコマンドの処理、詳細は略 */
            return(clscmd(dev, SCMD_UCMD, data));
            break;
        default:
            /*
             * Unknown ioctl
             */
            return(ENOTTY);
    }
    return(0);
}

```


SCSI vs IDE

西岡督太郎

低価格 AT コンパチ機と DOS/V の組み合わせにより、一時旗色が悪くなっていた PC-98 が、デスク・トップ型の最新機種の内蔵ハード・ディスク・インターフェースに IDE を採用しました。

■ AT 化する 98

AT の世界では、システムの規模を大きくするにつれて、内蔵 IDE ドライブに加えて、SCSI ホスト・アダプタを追加し、ハード・ディスクや CD-ROM、そして光磁気ディスク、テープ・ドライブなどを増設していくというのが一般的になりました。ところが PC-9801 は、これまでの主流が外付けの SCSI ハード・ディスクであったにもかかわらず、標準的ハード・ディスクのインターフェースを、それもこの時期に変更したのは、価格面や Windows 3.x など考慮した結果でしょうか。またひとつ 98 の AT 化が進んだように思います。

■ 拡張 IDE の実力は？

技術的な面からいえば、IDE は、旧来の AT 用 ST-506 ディスク・コントロール・プロトコルの制約を引きずっており、現時点ではハード・ディスク専用のインターフェースで、最大 528 M バイトのドライブを 2 台までしか接続できません。

現在、米 Western Digital 社を中心に、現状の IDE を拡張し、接続可能ドライブ数を 4 台に、またドライブ当たり容量を Macintosh の SCSI と同様の LBA (Logical Block Address) 方式により 8.4 G バイトまで増やすという新しい規格が検討されているようです。この新規格では、CD-ROM やテープ・ドライブなども接続できるようになるようです (表参照)。

IDE の入出力に際しての手順はシングル・タスク的であり、ドライブに対して一度コマンドを発行した

ら、そのコマンドの動作が完了するまで、動作中のものを含むいかなるドライブにも、新たなコマンドを発行することができません。

■ シングル向きの IDE・マルチに強い SCSI

それに対して SCSI では、より汎用的なインターフェースとなっており、使い方によっては小規模 LAN 的な構成も可能です。ホスト・コンピュータを 1 台とし、他はすべて外部記憶装置にしてしまうという使い方であれば、外部記憶装置を最大 7 台まで接続することができます。そして 1 台の外部記憶装置は LUN で指定される最大 7 台までのドライブを内蔵できます。また、各外部記憶装置もしくはドライブは独立して動作できるので、ある外部記憶装置 (ドライブ) にコマンドを発行し、その動作の完了を待たずに、他の外部記憶装置 (ドライブ) にコマンドを発行するというようなこともできます。このことは、そのバス調停機能ともあいまって、Unix や Windows NT などのマルチ・ユーザ OS との整合性がよいということでもあります。さらに、ハード・ディスクに比べて低速のメカをもつデバイス、たとえばプリンタやイメージ・スキャナなどをシステム内に導入する際にも有効に作用します。

■ 共存する SCSI と IDE

SCSI、IDE にかぎらず、インター

フェース・バス規格上のデータ転送スピードと、システム・バスを経由しユーザの手元にデータが届くまでの実効速度は、システム・バス上のウェイトの有無や CPU のクロックなどにより、必ずしも一致しません。それは、AT コンパチ機同士でも、また PC-98 を含めてもメーカーや機種による差異があります。

ユーザの立場では、「SCSI だから」、「IDE だから」ということではなく、体感スピードや操作性、コスト、メーカーのサポートなどをも含め、コンピュータが快適に使えることを真に求めていると思います。

本稿の表題は「SCSI vs IDE」となっていますが、これら二つのインターフェースは、それぞれの守備範囲がたしかに重なっている面もありますが、一方が他方を駆逐するというようなことではなく、システムの内容や規模に応じて役割を分担しつつ、共存しながら発展していくのではないのでしょうか。

参考文献

- 1) "IDE versus SCSI: Which disk interface is best?" Distributed Processing Technology
 - 2) 「技術者のためのハード・ディスク 100% 活用法」, 「トラ技コンピュータ」, 1993 年 9 月号
 - 3) 「最新 SCSI マニュアル」, 別冊インターフェース
 - 4) 「特集: SCSI, IDE のディスク増設再点検」, 「日経バイト」, 1993 年 11 月号
- にしおか・とくたろう 執筆

表 IDE と拡張 IDE

	IDE	拡張 IDE
接続可能ドライブ数	1 系列 2 台	2 系列 4 台
ドライブ最大容量	528 M バイト	8.4 G バイト (Logical Block Address 方式による)
接続可能デバイス	ハード・ディスクだけ	ハード・ディスク CD-ROM ストリーマ・テープなど
データ転送方式	ISA バス	ローカル・バス対応

第5章

98 & AT用SCSIユーティリティ

互換性チェックや解析に威力を発揮するデバッグ・ツールの機能と使い方

真樹 美智

Inquiry リストの出力/緒元リストの出力/機器間複写機能/セクタ・ダンプ/セクタ内エディット/シーク試験/リード試験などの機能があり、操作は、1文字のコマンドを入力する方式。PC-98の55/92ボードおよびAT互換機/ASPIに対応している。(編集部)

1

開発の動機

SCSIという共通仕様があるにもかかわらず、現実にはホストのファイル・システムの都合、設計時のバグや不備、ベンダ・ユニークの多用などの理由により非互換性が存在しています。データ転送速度が遅い、または容量が少ないなどターゲットの能力不足が原因で、目的とするシステムが破綻するのであれば、ホスト・コンピュータが故意にそのターゲットとの接続を拒むことは正当だろうと思います。そうでないことが原因でSCSI規格を採用しているターゲットが動作しないのは不健全でしょう。

筆者がSCSIハード・ディスクの非互換の問題に興味をもち始めたのは3年ほど前になります。はじめSCSIモニタを使って解析をしていましたが、モニタリングだけではわからないことが多いのに気づきました。ハード・ディスクによってはInquiryやMode Senseコマンドの直後にハングするものもあれば、立ち上がりは正常でも、その後に誤動作するものもあります。これらの原因を調べるために、直接SCSIのReadコマンドを発行してハード・ディスクのパーティション情報をダンプし、OSのファイル・システムの流れを解析する必要が生じました。ここでとりあげるユーティリティSU.EXEの開発目的はこれらを解析

するためにつくったものです。

「4.1 98と55&92ボード(p.80~)」ではSU.EXEを使ってハード・ディスクとSCSIコントロール・ボードとの互換性を説明しています。SU.EXEユーティリティで出力されるInquiryリストから同期転送、ワイド・バス、リンク、コマンドのキューイングなどのサポートの有無、緒元リストからは、UNIXなどハード・ディスクの緒元の必要なホスト・コンピュータに接続できるかどうかの手がかりとなる情報として、シリンダ、ヘッド、セクタ数などの緒元情報を得て互換性の説明をしました。またハード・ディスクのパーティション管理領域をセクタ内エディット機能を使って再編集することで各ビットの意味を解析し、セクタ・ダンプ機能を使って解析結果を説明しています。また追加機能として、ハード・ディスクやMO間のイメージ・コピー機能、ハード・ディスクの転送速度、シーク速度の実力値を測定する機能をもりこみました。

2

SU.EXEの機能概要

SU.EXEは『インターフェース』'93年1月号に発表したSUT.EXEに機能追加したものです。SUT.EXEでは、Inquiryリスト、緒元リストの出力、および機器間複写機能がありました。SU.EXEでは、それらの機能のほかにセクタ・ダンプ、セクタ内エディット、シーク試験、リード試験の機能を追加しました。また、対象機種もNECの55/92ボードばかりではなく、IBM PC/ATおよび互換機(以下、AT互換機と表記する)のSCSIホスト・アダプタであるAdaptec(ASPI仕様)にも対応しました。

SUT.EXE との違いは、操作をコマンド入力形式としたところです。起動後、まず ASPI のファイル・ハンドルをオープンしてみて、オープンできれば ASPI 仕様の SCSI BIOS をコールするようにし、オープンしなければ NEC の 55/92 ボードの SCSI BIOS をコールするようになっています。したがって、PC/AT には ASPI 仕様のボードとデバイス・ドライバ、PC-9801 には 55 ボードまたは 92 ボードが必要となります。なお、ASPI 仕様のボードでの検証は以下の環境で行いました(AT 互換機の場合は、これ以外の組み合わせ試験を行っていません)。

▶ 検証した PC/AT での環境

コンピュータ：IBM PS/55 Z

OS：IBM DOS Version J5.00/V

SCSI ボード：Adaptec AHA-1542B

デバイス・ドライバ：ASPI4DOS.SYS

▶ NEC PC-9801 の動作環境

PC-9801 本体

PC-9801-55/92/または互換 SCSI ボード

注意) PC-9801-55 ボードは、ハード・ディスクに関して Inquiry データの Vendor Identification が 'NEC' でなければ永久ループし、システムが立ち上がりません。また 55 互換ボードのなかにもハード・ディスクが同一メーカーでないときにシステムが立ち上がらないボードもあります。この場合、以下の手順で SU.EXE を起動します。

- (1) ハード・ディスクの電源を落とす
- (2) システムを立ち上げる
- (3) ハード・ディスクの電源を入れる
- (4) ユーティリティの起動

■ SU.EXE の機能

SU.EXE の機能を列記すると以下のようになります。

- (1) 緒元リストの表示
- (2) Inquiry リストの表示
- (3) ハード・ディスク、光磁気ディスク間イメージ・コピー
- (4) セクタ/ダンプ
- (5) シーク・タイム・テスト
- (6) リード・タイム・テスト

3

SU.EXE の使用方法

▶ 書式

書式はつぎのようになります。

SU <SCSI ID>

<SCSI ID>は、目的とする SCSI 機器の ID のことです。0 から 6 の範囲で指定します。

▶ 起動例

A>SU 1

SU Version 1.1 (C) Copyright AUG 1993 by Y.
MAKI

起動直後、入力促進のプロンプト '!' の表示がされます。プロンプト '!' の後に実行したいコマンド(パラメータ)を入力します。

▶ コマンド・ヘルプ

コマンド・ヘルプ '?' を入力すると図 1 のようなコマンド一覧が表示されます。'Q' を入力すると OS にもどります。

4

SU.EXE のコマンド 詳細

■ G コマンド(緒元リスト)

たとえば、ID 0 のハード・ディスクのジオメトリ・リストがほしいとしましょう。その場合には、以下のように入力すると図 2 のような緒元リストが得られます。

A>SU 0

SU Version 1.1 (C) Copyright AUG 1993 by Y.
MAKI

-G

● 緒元データの説明

G コマンドで得られた図 2 の緒元リストの各項目について説明しましょう。

(1) バイト容量：論理装置の容量のことで、Read Capacity コマンドで得た論理ブロック数と論理プロ

図1 コマンド・ヘルプ「?」を入力すると、コマンド一覧が表示される



ック・サイズを乗算した値をキロバイトになおした表示としました。

(2) 論理ブロック数：Read Capacity コマンドで得た論理ブロック・アドレス(0 から起算した最終ブロック・アドレス)+1 した値です。

(3) 論理ブロック・サイズ：Read Capacity コマンドで得た論理ブロック長です。物理セクタ・サイズと違うことがあります。

(4) シリンダ数/ヘッド数：Mode Sense コマンドのページ 4 (Rigid Disk Drive Geometry Page) から得たシリンダ数、ヘッド数です。

(5) セクタ/トラック：以降サーフェスまでは、Mode Sense コマンドのページ 3 (Format Device Page) で得た情報です。トラック当たりのセクタ数のことで、交代セクタがふくまれることがあります。

(6) トラック/ゾーン：ゾーンを単位として交代セクタ、および交代トラックを割り当てる目的で使用され、単位はトラックです。この値が 0 のとき装置全体をゾーンとみなします。

(7) 交代セクタ/ゾーン：ゾーン中の交代セクタ数で、Format Unit コマンド実行中アドレス可能ブロックから除外され、不良セクタの交代用に使用されます。

図2 緒元リスト-SCSI ID 0 のハード・ディスクのジオメトリ・リストを出力



(8) 交代トラック/ゾーン：ゾーン中の交代トラック数で、Format Unit コマンド実行中アドレス可能ブロックから除外され、不良セクタの交代用に使用されます。

(9) 交代トラック/論理ユニット：論理ユニット当たりの交代トラック数で、Format Unit コマンド実行中、アドレス可能ブロックから除外され、不良セクタの交代用に使用されます。

なお、交代セクタ/ゾーン、交代トラック/ゾーン、交代トラック/論理ユニットのフィールドの値が 0 のとき、これらの値はターゲット自身が決定することを示します。

(10) インタリーブ・ファクタ：ホストのバス転送速度と、ハード・ディスクからのデータ転送速度との調和をとるため、物理ブロックと論理ブロックとの関係を示す値です。インタリーブの値を N としたとき、 $N-1$ の物理ブロックの間隔で論理ブロックが割り付けられます。インタリーブ 1 の値は連続した論理ブロックが物理ブロックと一致していることを表します。これは、連続したブロック・アクセスをする場合、読み込み/書き込みヘッドのポジションがთვისの論理ブロックにくるように働き、ディスクの回転待ちによる無駄な時間を防ぎます。最大インタリーブ $\langle N_{max} \rangle$ は次式

で求めます。

$\langle N_{\max} \rangle = \text{セクタ/トラック} - \text{交代セクタ/トラック} - 1$

インタリーブ2の例を表1に示します。

- (11) トラック・スキュー・ファクタ：同一シリンダ中において、あるトラックの最後の論理ブロックと、つぎのトラックの最初の論理ブロックの間の、物理ブロック数のことです。セクタ・インタリーブ使用時に、トラックにまたがる連続アクセスをしようとした場合、ヘッド切り替え時の回転待ち時間を最小にする意味があります。
- (12) シリンダ・スキュー・ファクタ：シリンダの最終論理ブロックと、つぎのシリンダの最初の論理ブロック

表1 インタリーブ2の例

物理ブロック・アドレス	0	1	2	3	4	5	6...
論理ブロック・アドレス	0	18	1	19	2	20	3...

の間の物理ブロック数のことで、シリンダ・シーク・タイムに、この値を対応させることで回転待ち時間を防ぎます。

(13) ソフト・セクタ方式：このビットがオン(YES)の場合、ターゲットはソフト・セクタ・フォーマティングを使用することを示します。

(14) ハード・セクタ方式：このビットがオン(YES)の場合、ターゲットはハード・セクタ・フォーマティングを使用することを示します。

(15) リムーバブル：このビットがオン(YES)の場合、メディアは取り外し可能であることを示します。Inquiry データ中にも同様なビットがあります。

(16) サーフェス：このビットは装置が論理ブロックをどのようにアドレッシングするかを示します。オンの場合、つぎのシリンダに論理ブロックを割り付ける以前に、シリンダ内の全セクタに論理ブロックを割り付けることを意味します。当ユーティリティでは「シリンダ順」の表示をします。オフの場合、サーフェス(面)すべ



UNIX のデバイス・ファイル

パソコン用のハード・ディスクでは、それほど問題にならないシリンダ、ヘッド、トラックあたりのセクタ数なども UNIX に組み込もうとする場合には必要不可欠な情報といえます。UNIX の場合、ハード・ディスクの緒元をデバイス・ファイルといわれるテキストに書き込みます。これはパーティションをシリンダ単位にするための情報です。余談ですが、シリンダ当たりの容量が大きいほど、ファイル・アクセス時のシーク回数を減らすことができます。この意味からするとシリンダ方向に深い、つまりヘッド数が多いのもあながち無意味でもないでしょう。最近のハード・ディスクはシーク時間も短くなっていますが、薄型でヘッド数が少なく、シリンダ数、トラックあたりのセクタ数が多くなってきているようです。

この例は NEWS(NEWS-1460, NEWS-OS 3.3 a) に図2のハード・ディスクを登録したときのものです。機種によって多少の違いがあります。UNIX によっては、デバイス・ファイルに回転数などを書き込むものもあります。UNIX でのデバイス・ファイルの例を図Aに示します。

図A UNIX でのデバイス・ファイルの例

```
type: SCSI
disk: DRR100D
label:
flags:
bytes/sector: 512
sectors/track: 35
tracks/cylinder: 4
sectors/cylinder: 140
cylinders: 1465
rpm: 3600
interleave: 1
trackskew: 0
cylinderskew: 0
headswitch: 0 # milliseconds
track-to-track seek: 0 # milliseconds
drivedata: 0
```


てに論理ブロックを割り付けた後、つぎのサーフェスの割り付けに移ることを意味します。SU.EXE では「ヘッド順」の表示をします。

(17) 論理構成：ハード・ディスクを UNIX に組み込む際、諸元情報が必要になりますが、ハード・ディスクによっては論理ブロック・サイズと物理セクタ・サイズの違い、トラック中に交代セクタがあるなど、このままのデータでは UNIX のデバイス・ファイル(コラム参照)に登録できないことが理解できると思います。ここで使っている論理構成とは、いままでのデータを元に筆者なりに論理的なシリンダ、ヘッド、セクタ/トラック、ブロックを試算した値です。もしも論理構成のブロックの数値がリスト中の論理ブロック数と一致すれば、論理構成の数値をデバイス・ファイルにこのまま登録できるはずですが、論理構成のブロックの数値がリスト中の論理ブロック数より少ない場合、有効データが1シリンダ(または数シリンダ)中におさまるように論理構成を変えてみてください。

つぎに ID 1 の MO ディスクのジオメトリ・リストを出力してみます(図 3)。MO の場合、ハード・ディスクのように Mode Sense コマンドでページ 3、4 を得ることはできませんので、Check Condition のステ

ータスを返してきます。このとき Request Sense コマンドで得た結果のセンス・キーの内容は Illegal Request で、追加センス・コードの内容は Invalid Field In CDB となります。なお、Read Capacity コマンドは正常に受け付けられたので、返り値を表示します。

■ I コマンド(Inquiry リスト)

図 4 に、Inquiry リストの出力例を示します。

■ C コマンド(機器間複写機能)

この機能は、Read Capacity コマンドで送り側、受け側のハード・ディスクの容量を調べ、等しいか、少ないほうの容量で転送を行います。また異なる論理ブロック・サイズでも、論理ブロックを転送データ量に合わせて複写します。じつのところハード・ディスク(または MO)間の複写と書きましたが、Inquiry コマンドを発行しての周辺機器種別はチェックしていないので、受け側が書き込み不可媒体(ライト・プロテクトを含む)でなければ、書き込めるはずですが、もし、なんらかの原因でアクセスできなければ Check Condition のステータス(センス・キーの内容)を返し、処理を中断します。

図 3 SCSI ID 0 の MO のジオメトリ・リストを出力

```
A>SU 1
SU Version 1.1 (C) Copyright AUG 1993 by Y.MAKI
-G
----- 緒元データ -----
バイト容量      : 127393 キロバイト
論理ブロック数  : 248816
論理ブロックサイズ : 512 バイト
物理セクタサイズ : 不明
シリンダ数      : 不明
ヘッド数        : 不明
セクタ/トラック : 不明
トラック/ゾーン : 不明
交代セクタ/ゾーン : 不明
交代トラック/ゾーン : 不明
交代トラック/論理ユニット : 不明
インターリーブファクタ : 不明
トラックスキューファクタ : 不明
シリンダスキューファクタ : 不明
ソフトセクタ方式 : 不明
ハードセクタ方式 : 不明
リム・ハブ      : 不明
サーフェス      : 不明
ID 3 が ILLEGAL REQUEST の センシキを返してきました
```

図 4 SCSI ID 0 の Inquiry リストを出力

```
A>SU 6
SU Version 1.1 (C) Copyright AUG 1993 by Y.MAKI
-I
----- INQUIRYデータ -----
装置機種        : 磁気ディスク
リム・ハブ      : NO
ISOバージョン   : 0
ECMAバージョン  : 0
ANSIバージョン  : 1
データ書式      : SCSI-1
相対アドレス指定 : NO
32ビットバス    : NO
16ビットバス    : NO
同期モード      : NO
コマンドリンク  : NO
コマンドキュー  : NO
ソフトセクタ    : NO
製造元          : ALPS
装置名          : DRR100D
装置バージョン   : 1.B
```


バックアップで使う際、受け側の容量は等しいか大きい必要があります。送り側のハード・ディスクのデータにトラブルがあった場合、受け側から戻すことにより、まったく同じ内容で復帰します。また、ハード・ディスクにインストールされている OS の種別を問わず、同じ内容のハード・ディスクを何台も作るのにも役立ちます。

図 5 に、SCSI ID 0 から ID 3 への複写の例を示します。起動時に指定された SCSI ID 0 から ID 3 にイメージ・コピーを行うものです。

図 5 の例の転送ブロック・サイズは少ないほうのデータ容量のブロック・サイズを表示します。(Y/N)? の入力促進に対して 'y' を入力することで複写が開始され、下の行に終了した論理ブロック・アドレスが表示されます。

■ 受け側ハード・ディスク扱い上の注意

受け側ハード・ディスクをたんにバックアップ専用として扱い、送り側の不慮の事故のとき戻すだけに使うのであれば、問題はありませんが、バックアップしたハード・ディスクをそのまま使う場合は、注意が必要です。PC-9801 で使うのであれば、送り側のハード・ディスクと、まったく同一のドライブであることと、インターフェース・ボードも同じでなければいけません。同一ドライブであってもインターフェース・ボードを違えることにより、論理的な構成を変えられる可能性があり、その結果、とんでもないところにアクセスされることがあるからです。

Macintosh の場合、受け側が大きければパーティションも論理ブロックの若い順で区切られているので動作するはずですが、しかし、インストーラ・ソフトでそ

のハード・ディスクに最適なインタリーブ値(可変であれば)でフォーマットされていないのであれば、バックアップする前に物理フォーマットを施したほうが賢明といえます。インタリーブ値によってはアクセス速度が倍違うことがあるからです。

FMR, Panacom M の場合、パーティションを論理ブロックの大きいほうから区切っていきます。したがって、受け側ハード・ディスクを送り側より少ない容量にし、送り側のパーティション容量を受け側より少なく区切ったつもりでそのまま使おうとしても、最初に区切ったパーティションにデータが入っていないことになります。

■ セクタ・ダンプ

セクタ・ダンプの入力書式はつぎの三つです。

(1) -d

ロジカル・ブロックを指定しない場合、起動直後のロジカル・ブロックは 0 です。その後、ダンプ操作をこの書式で行ったとき、前回表示されたロジカル・ブロック+1 のブロックが表示されます。表示例を図 6 に示します。

(2) -d[ロジカル・ブロック番号]

指定されたロジカル・ブロック番号を 1 ブロック表示します。

(3) -d[開始ロジカル・ブロック番号],[終了ロジカル・ブロック番号]

開始ロジカル・ブロック番号から終了ロジカル・ブロック番号までの表示を行います。

■ セクタ内エディット

セクタ内エディットは指定したセクタ内の内容を書き換えることです。筆者は、パーティション構造を調べるときに、あるビットをオン/オフすることで、そのビットの意味を知るようにしています。この機能を使う場合、あらかじめ該当するブロックのセクタ・ダンプをリダイレクションなどでとってから行ってください。変更内容によっては、システムが立ち上がらなくなる恐れもあります。

・書式

E[<タイプ>]<ブロック>,<オフセット>[,<リスト>]

<タイプ> Byte, Word, Ascii の頭文字を入力する。省略時は Byte。

<ブロック>,<オフセット>:編集するブロック内の

図 5 SCSI ID 0 から ID 3 への複写の例

```
A>SU 0
SU Version 1.1 (C) Copyright AUG 1993 by Y. MAXI
-C3
転送ブロックサイズ=205100
ID 0(205100:512) から ID 3(248816:512) へ 転送(Y/N) y
205100
-
```

<説明>上の例の転送ブロック・サイズは少ないほうのデータ容量のブロック・サイズを表示する。(Y/N)? の入力促進に対し、'y' を入力することで複写が開始され、下の行に終了した論理ブロック・アドレスが表示される。

図6 表示例：PC-9801用SCSIハード・ディスクの論理ブロック0

表示例(NEC PC-9801用SCSIハードディスクの論理ブロック0)

```
-d
0000 EB 0A 90 90 49 50 4C 31-00 00 00 1E A0 84 05 B4 .....IPL1.....
0010 8E CD 1B A8 20 74 31 32-DB B4 14 CD 1B 72 29 80 .....t12.....r).
0020 FB 84 75 24 B4 B0 BE D7-59 BA 06 00 1E 0E 1F CD .....uS....Y.....
0030 1B B4 B0 CD 1B 1F 73 03-EB 6B 90 B4 24 BB 00 04 .....s..k..S...
0040 B9 30 12 BA 40 01 CD 1B-B8 00 01 B4 84 CD 1B B4 .....0..8.....
0050 06 33 C9 33 D2 50 8C C8-2D 40 02 8E C0 58 33 ED .....3..P...-8...X3.
0060 CD 1B 72 41 B4 06 BA 01-00 81 C5 00 04 CD 1B 72 .....rA.....r
0070 34 BA 04 00 81 FB 00 02-75 03 BA 02 00 B4 06 BB .....4.....u.....
0080 00 1C 81 C5 00 04 CD 1B-72 1B 50 8B C5 B1 04 D3 .....r..P.....
0090 E8 8C C1 03 C1 8B F0 58-E8 15 00 2E 89 36 0A 00 .....X.....6..
00A0 2E FF 1E 08 00 E8 08 00-B4 0E CD 1B B9 01 00 CB .....V...2....r...u
00B0 56 A0 84 05 32 DB B4 14-CD 1B 72 1D 80 FB 84 75 .....Y...Y....
00C0 1B B4 B0 2E C6 06 DB 59-00 BE D7 59 BA 06 00 1E .....
00D0 0E 1F CD 1B B4 B0 CD 1B-1F 5E C3 1E 00 00 00 01 .....
00E0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
00F0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....U.
0100 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0110 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0120 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0130 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0140 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0170 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0180 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0190 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
01A0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
01B0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
01C0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
01D0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
01E0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
01F0 00 00 00 00 00 00 00 00-00 80 00 08 00 55 AA .....U.
```

図7 セクタ内エディットの例

```
-d237704
0000 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0010 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0020 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0030 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0040 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
01B0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
```

(a) 237704 ブロックの内容

```
-e 237704 0 .....Byte(省略時)での編集
0237704:0000[000] 00'.1
0237704:0001[001] 00'.2
0237704:0002[002] 00'.3
0237704:0003[003] 00'.4
0237704:0004[004] 00'.5
0237704:0005[005] 00'.6
0237704:0006[006] 00'.7
0237704:0007[007] 00'.8
0237704:0008[008] 00'.
(サ-ツで編集終了)
7' 0000(237704)の内容を変更しますか?(Y/N)y
-d237704
0000 01 02 03 04 05 06 07 08-00 00 00 00 00 00 00 .....
0010 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
```

(b) 操作例

図8 Wordでの編集の例

```
-ew 237704 0 .....Wordでの編集
0237704:0000[000] 0000.1234
0237704:0001[001] 0000.5678
0237704:0002[002] 0000.9abc
0237704:0003[003] 0000.def0
0237704:0004[004] 0000.
0237704:0005[005] 0000.
7' 0000(237704)の内容を変更しますか?(Y/N)y
-d237704
0000 54 12 78 56 BC 9A F0 DE-00 00 00 00 00 00 00 .....4.xV.....
0010 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
```

図9 Asciiでの編集の例

```
-d 237704
0000 31 32 33 34 35 36 37 38-39 00 00 00 00 00 00 .....123456789.....
0010 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....

-ew 237704 0 .....Asciiでの編集
0237704:0000[000] '1' 31.i
0237704:0001[001] '2' 32.n
0237704:0002[002] '3' 33.t
0237704:0003[003] '4' 34.e
0237704:0004[004] '5' 35.r
0237704:0005[005] '6' 36.F
0237704:0006[006] '7' 37.a
0237704:0007[007] '8' 38.c
0237704:0008[008] '9' 39.o
0237704:0009[009] '. ' 00.
7' 0000(237704)の内容を変更しますか?(Y/N)y
-d 237704
0000 49 6E 74 65 72 46 61 63-65 00 00 00 00 00 00 .....InterFace.....
0010 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
```

オフセットを指定する。省略不可。

<リスト>: <リスト>を省略した場合、<タイプ>ごとの入力促進がある。入力促進はリターンで打ち切れる。

仮に 237704 ブロックに図7(a)の内容があるとしたときの操作例を図7(b)に示します。

同様の結果がつぎのコマンドでも得られます。

-e 237704 0 1 2 3 4 5 6 7 8(リストを使用した場合1から8がリスト)

ブロック(237704)の内容を変更しますか?(Y/N)y

同じエリアをワードでうめる場合は、図8のようにします。同様の結果がつぎのコマンドでも得られます。

-e 237704 0 1234 5678 aabc defg

ブロック(237704)の内容を変更しますか?(Y/N)y

同じエリアをASCIIでうめる場合は、図9のようにします。同様の結果がつぎのコマンドでも得られます。

-e 237704 0 InterFace

ブロック(237704)の内容を変更しますか?(Y/N)y

■ シーク・タイム・テスト

ご存知のようにシークとは、リード・ライト・ヘッ

ドが目的とするトラックに移動することです。しかしながら SCSI の Seek コマンドには、LBA(ロジカル・ブロック・アドレス)の指定のみで、トラックの指定はありません。正確にシーク速度を測定するには、現在のトラック位置からみて、リード・ライト・ヘッドが何トラック移動するのかを、論理的なシリンダ数、ヘッド数、トラックあたりのセクタ数から目的の LBA を算出し、コマンドを発行する必要がありますが、さきほどの互換性問題のところで述べたように、Mode Sense データのページ 3 には、交代トラックや、交代セクタの存在、論理ブロック・サイズと物理ブロック・サイズの違いなどで、ページ 4 のシリンダ数をそのまま使うことができません。また、これらの数値を電卓ではじくことも、たいへん面倒なことです。

SU.EXE では、このあたりの抜け道としてシーク試験では、あらかじめ試算しておいた論理的なシリンダ数、ヘッド数、トラックあたりのセクタ数(論理構成：図 2 参照)から目的の LBA を算出し、だいたいのシリンダ方向へのヘッドの移動がわかるようにしたつもりです。ここで「だいたいの」といったのは、Read Capacity コマンドで得た論理ブロック数と、論理構成のブロック数が一致しないとき、シーク試験の正確さは、「だいたいの」という表現がふさわしいと思うからです。これは筆者の感想ですが、世に出ている SCSI ハード・ディスクを調べてみて、Mode Sense データのページ 3, 4 からシリンダ数、ヘッド数、トラックあたりのセクタ数から算出した論理ブロック数と、Read Capacity コマンドで得た論理ブロック数との一致が、むずかしいことを痛感します。SCSI でのアクセスの仕方が論理ブロック・アドレスとの理由からか、どうも正直に記述(ページ 3, 4 が)されていないようなハード・ディスクもあるようです。

▶ シーク・タイム・テストでのプログラム動作

シーク速度を正確に測定するには、前述のシリンダ数、ヘッド数、トラックあたりのセクタ数から目的の LBA を算出し、SCSI コマンドを発行します。コマンドが発行されてから、アービトレーション、セレクション、コマンド・インの各フェーズの後、シークが起こり、ステータス、メッセージ・インでプログラムに戻ってきます。実際のシーク速度は、ヘッドの移動時間ですから、コマンドの発行から、戻ってくるまでのトータル時間のなかから、シーク時間だけを抜き出す

必要があります。

このユーティリティでは、まず同じトラック位置(同じ LBA)に Seek コマンドを 1000 回(無指定時)発行します。こうすることで実際のシーク動作を起こさない秒数(コマンド発行から戻りまで)を計測しておきます。

つぎに、0 シリンダ目から全シリンダの 3 分の 1 のシリンダ位置にシークすることを 1000 回繰り返す、実際のシーク動作の起こった時間(トータル時間)を計測します。この時間から、さきほどのシーク動作をともなわない時間を引くことでシーク時間(平均シーク・タイム)を求めました。

なお、検査前にドライブが、ハード・ディスクか MO(光磁気ディスク)かを調べ、それら以外のドライブについては、検査しません。MO については、ヘッド数 1、トラックあたりのセクタ数 25 とし、シリンダ数は、Read Capacity コマンドで得た論理ブロック数を 25 で除算した値としました(図 10)。

■ リード・タイム・テスト

最近では、ハード・ディスクの大容量化にともない、キャッシュ・メモリを内蔵しているドライブも増えてきました。なかには 1 M バイトを超えるキャッシュ・メモリを搭載しているものもあります。リードの検査をするうえで、注意しなければならないことは、キャッシュ・メモリにヒットさせずに、実際の盤面からデータを読み込むことでハード・ディスクの実力値を測らなければならないことです。

もしも、目的の論理セクタからアロケーション長までのデータがドライブ内のキャッシュ・メモリにあれば、データの経路はメモリと SCSI バス間だけとなり、肝心の読み取り速度がわからなくなります。こうした現象は、試験データが少なく、試験回数が多く、かつキャッシュ・メモリが大きい場合、顕著にあらわれます。

SU.EXE では、これを回避するため、前もって 2 M バイトまでのシーケンシャル・データを読み込み、キャッシュ・メモリを満杯とし(2 M バイト以内であれば)、その後キャッシュ・メモリにリードされていないセクタからシーケンシャルに 32 K バイトずつのリードを行っています。ここで「ハード・ディスクの実力値」と書きましたが、ハード・ディスクの同期転送の機能がないか、あるいはハード・ディスクに同期転送の能

図10 シーク・タイム・テストの例

```
-s ...①
シリンダ 0-488 (LBA:68320) 間で1000回シーク(Y/N)?y
オーバーヘッド 検査のシーク開始 シーク回数:1000 ...②
オーバーヘッド 検査のシーク秒数:2 ...③
シーク開始 シーク回数:1000 ...④
シーク秒数:23 ...⑤
シーク秒数からオーバーヘッド 秒数を引いた秒数:21 ...⑥
平均シークタイム:21ms ...⑦
-
```

<説明>

① 省略時 1000 回のシーク・コマンド 回数を指定したければ 'S' のあとに 10 進数で最大 32767 まで指定できる。計測が秒単位なので、大きめの数値がより正確になる。計測してみても、平均シーク・タイムが前後するようならば、安定するまで大きくする。'y' で計測が開始される。

例: -s3000

- ② シーク動作を伴わない Seek コマンド発行の開始
- ③ シーク動作を伴わない Seek コマンドの経過秒数
- ④ シーク動作を伴う Seek コマンド発行の開始
- ⑤ シーク動作を伴う Seek コマンドの経過秒数
- ⑥ ⑤から③を引いた秒数
- ⑦ ⑥を 1000 で割った時間(平均シーク・タイム)

図11 リード・タイム・テストの例

```
-r ...①
16メガバイトのリード (Y/N)?y
オーバーヘッド のシーク開始 ...②
オーバーヘッド のシーク秒数:4 ...③
キャッシュ・メモリへの読み込み(2Mバイト)開始 ...④
リード開始 ...⑤
リード 秒数:64 ...⑥
リード 秒数からオーバーヘッド 秒数を引いた秒数:60 ...⑦
リード 転送レート256キロバイト/秒 ...⑧
-
```

<説明>

① 省略時 512 回のリード・コマンド: 回数を指定したければ 'R' のあとに 10 進数で最大 32767 まで指定できる。計測が秒単位なので、大きめの数値がより正確になる。計測してみても、リード転送レートが前後するようならば、安定するまで大きくしてみる。'y' で計測が開始される。

例: -r1000

- ② Seek コマンド発行の開始
- ③ Seek コマンドの経過秒数
- ④ キャッシュ・メモリへ 2 M バイトの読み込み開始
- ⑤ Read コマンド発行の開始
- ⑥ Read コマンドの経過秒数
- ⑦ ⑥から③を引いた秒数
- ⑧ 転送データ量を⑦で割ったリード転送レート(キロバイト/秒)

力があっても、ホスト・アダプタにその機能がない場合もあります。また、双方に同期転送の機能があっても、たがいの能力差が原因で、同期のネゴシエーション時にどちらかが断ることもできます。

カタログなどのスペックには、最大能力が書かれています。その能力をできるだけ引き出すには、SCSI モニタなどで通信の流れを調べる必要があります。

▶ リード・テストでのプログラム動作

リード・テストもシーク・テストの方法のところで述べたようにオーバーヘッド分を全体の計測時間から除く必要があります。このため試験の前半で、Read コマンドで指定する論理セクタと同じ LBA に対し Seek コマンドを発行し、試験回数分の時間を測っておきます。その後、Read コマンドで試験を行います。Read

コマンドでの時間から Seek コマンドの時間を引くことで読み込みにかかった時間を抜き出すことができます。

無指定での試験回数は 512 です。1 回のリード・サイズが 32 K バイトですから、全体で 16 M バイトを読み出すことになります。なお、Read コマンドでの論理セクタとアロケーション長がハード・ディスクの最大容量を越えそうになると、論理セクタが 0 に復帰して試験を継続するようにしました。試験を中止したい場合、ESC キーを押します(図11)。

参考文献

- 1) SCSI-2 X3T9.2/86/109, ANSI
- 2) 「SCSI インターフェーステクニカルブック」, ㈱コーラル

まき・よしとも

好評発売中

オープン・システム実践教室

事例に学ぶマルチベンダ・システム構築法 小暮裕明編著 定価1,650円

ホスト接続 LAN・WAN から PC-LAN の OA システムまでを体系づけて詳細に解説!

CQ出版社


```

/*
 * SU.C
 *
 * 機能1: ハードディスクから 諸元情報を得る
 * 機能2: ハードディスクInquiry情報を得る
 * 機能3: ハードディスクまたはMO間の複写
 * 機能4: セクタ・ジャンプ
 * 機能5: セクタ・ステップ
 * 機能6: シーク速度試験
 * 機能7: リード・転送レート試験
 *
 * 発行するコマンド:
 *     Test Unit Ready
 *     Inquiry
 *     Read Capacity
 *     Mode Sense(page 3 & 4)
 *     Read
 *     Write
 *     Seek
 *     Request Sense
 *
 * 使い方:
 *     SU <SCSI ID>
 *           <SCSI ID>: 目的SCSI装置
 *
 * Copyright Aug 1993 Y.M
 */
#include <stdio.h>
#include <dos.h>
#include <memory.h>
#include <malloc.h>
#include <string.h>
#include <time.h>
#include <conio.h>
#include <stdlib.h>
typedef unsigned long   ulong;
typedef unsigned short ushort;
typedef unsigned char   uchar;

#define TRUE            1
#define FALSE          0
#define YES             1
#define NO              0
#define FAIL            (-1)
#define NORMAL          0
#define ERROR           (-2)
#define SCSIBIOS        0x1b
#define CR              0x0d
#define LF              0x0a
#define ESC             0x1b
#define BUFF_SIZE       0x8000
#define SEEK_COUNT      1000
#define READ_COUNT      512
/*----- SCSI Status */
#define STS_GOOD         0x00
#define STS_CHK_CONDITION 0x02
#define STS_BUSY        0x08
#define STS_ERROR       (-3)
#define TIMEOUT         0x11
/*----- SCSI Phase */
#define DATA_IN_PHASE   0x08
#define DATA_OUT_PHASE  0x10
#define NO_TRANSFER      0x18
/*----- ASPIのための構造体 */
struct Request_Block {
    uchar Command_Code; /* W */
    uchar Status; /* R */
    uchar Host_Adapter_Number; /* W */
    uchar SCSI_Request_Flags; /* W */
    uchar Reserved[4]; /* - */
};

struct Host_Adapter_Inquiry {
    struct Request_Block rblock;
    uchar Number_of_Host_Adapters; /* R */
    uchar ID_of_Host_Adapter; /* R */
    uchar SCSI_Manager_ID[16]; /* R */
    uchar Host_Adapter_ID[16]; /* R */

```

```

    uchar Host_Adapter_Unique_Parameters[16]; /* R */
} ha_inquiry;

struct SCSI_IO_Req {
    struct Request_Block rblock;
    uchar Target_ID; /* W */
    uchar LUN; /* W */
    long Data_Allocation_Length; /* W */
    uchar Sense_Allocation_Length; /* W */
    uchar *Data_Buffer_Pointer; /* W */
    uchar *SRB_Link_Pointer; /* W */
    uchar SCSI_CDB_Length; /* W */
    uchar Host_Adapter_Status; /* R */
    uchar Target_Status; /* R */
    void (*POST_Routine)(); /* W */
    uchar Reserved[34]; /* - */
    uchar CDB[76]; /* R */
} scsi_io;

/*-----*/
struct SCSPACKET {
    uchar cdb_length;
    uchar *cdb;
    uchar req_flags;
    unsigned data_length;
    uchar *buffer;
    uchar status;
    uchar message;
    uchar *sense;
};

union REGS rsi;
union REGS cpu;
struct SREGS seg;
unsigned data_seg;

uchar work[256];
uchar sense_buffer[36];

uchar cdb_00[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
uchar cdb_03[] = {0x03, 0x00, 0x00, 0x00, 0x16, 0x00};
uchar cdb_08[] = {0x08, 0x00, 0x00, 0x00, 0x00, 0x00};
uchar cdb_0a[] = {0x0a, 0x00, 0x00, 0x00, 0x00, 0x00};
uchar cdb_0b[] = {0x0b, 0x00, 0x00, 0x00, 0x00, 0x00};

uchar cdb_12[] = {0x12, 0x00, 0x00, 0x00, 0xff, 0x00};
uchar cdb_1a[] = {0x1a, 0x00, 0x03, 0x00, 0xff, 0x00};
uchar cdb_25[] = {0x25, 0x00, 0x00, 0x00, 0x00, 0x00,
                  0x00, 0x00, 0x00, 0x00};

uchar *read_lba_pointer = cdb_08+1;
uchar *read_blk_pointer = cdb_08+4;
uchar *write_lba_pointer = cdb_0a+1;
uchar *write_blk_pointer = cdb_0a+4;
uchar *seek_lba_pointer = cdb_0b+1;
uchar *pgcode_pointer = cdb_1a+2;

struct SCSPACKET test_unit =
    {6, cdb_00, NO_TRANSFER, 0L, work, 0, 0, sense_buffer};

struct SCSPACKET request_sense =
    {6, cdb_03, DATA_IN_PHASE, 22L, sense_buffer, 0, 0, sense_buffer};

struct SCSPACKET readhd =
    {6, cdb_08, DATA_IN_PHASE, 0L, work, 0, 0, sense_buffer};

struct SCSPACKET writehd =
    {6, cdb_0a, DATA_OUT_PHASE, 0L, work, 0, 0, sense_buffer};

struct SCSPACKET seekhd =
    {6, cdb_0b, NO_TRANSFER, 0L, work, 0, 0, sense_buffer};

struct SCSPACKET inquiry =
    {6, cdb_12, DATA_IN_PHASE, 0xffL, work, 0, 0, sense_buffer};

struct SCSPACKET mode_sense =
    {6, cdb_1a, DATA_IN_PHASE, 0xffL, work, 0, 0, sense_buffer};

struct SCSPACKET read_capa =
    {10, cdb_25, DATA_IN_PHASE, 8L, work, 0, 0, sense_buffer};

```



```

ulong  lblock_total, logic_blocks, logic_cyl;
unsigned logic_sec_trk, lblock_len, heads;
char    #s[17], Token[80][80];
long    err_block = 0L;
uchar   device_type;

char    s_device_type[28];
char    s_rmb[8];
char    s_iso_ver[8];
char    s_ecma_ver[8];
char    s_ansi_ver[8];
char    s_dform[8];
char    s_reladr[8];
char    s_wbus32[8];
char    s_wbus16[8];
char    s_sync[8];
char    s_linked[8];
char    s_cmdque[8];
char    s_sftre[8];
char    s_vend_id[12];
char    s_prod_id[20];
char    s_prod_rev[10];

char    #ss_inquiry[] = {
    "----- INQUIRY -----", "Yn",
    "装置機種      : ", s_device_type,
    "リム-ハーフ  : ", s_rmb,
    "ISOハーフ  : ", s_iso_ver,
    "ECMAハーフ  : ", s_ecma_ver,
    "ANSIハーフ  : ", s_ansi_ver,
    "フォーマット : ", s_dform,
    "相対アドレス指定: ", s_reladr,
    "32ビットハーフ : ", s_wbus32,
    "16ビットハーフ : ", s_wbus16,
    "同期モード   : ", s_sync,
    "コマンドリンク : ", s_linked,
    "コマンドキュー : ", s_cmdque,
    "ソフトリセット : ", s_sftre,
    "製造元       : ", s_vend_id,
    "装置名       : ", s_prod_id,
    "装置バージョン : ", s_prod_rev,
    ""};

char    s_lblock_total[16];
char    s_lblock_len[24];
char    s_capacity[24];
char    s_track_zone[8];
char    s_alsec_zone[8];
char    s_altrk_zone[8];
char    s_altrk_lunit[10];
char    s_sec_track[8];
char    s_pblock_len[24];
char    s_interleave[8];
char    s_trk_skew[8];
char    s_cyl_skew[8];
char    s_soft_sec[8];
char    s_hard_sec[8];
char    s_surface[16];
char    s_cylinder[10];
char    s_heads[8];

char    #ss_geometry[] = {
    "----- 緒元 -----", "Yn",
    "バイト容量      : ", s_capacity,
    "論理ブロック数  : ", s_lblock_total,
    "論理ブロックサイズ : ", s_lblock_len,
    "物理ブロックサイズ : ", s_pblock_len,
    "シリンダ数      : ", s_cylinder,
    "ヘッド数        : ", s_heads,
    "セクタ/トラック : ", s_sec_track,
    "トラック/ゾーン : ", s_track_zone,
    "交代セクタ/ゾーン : ", s_alsec_zone,
    "交代トラック/ゾーン : ", s_altrk_zone,
    "交代トラック/論理ユニット : ", s_altrk_lunit,
    "インターリーブ方式 : ", s_interleave,
    "トラックスキュー方式 : ", s_trk_skew,
    "シリンダスキュー方式 : ", s_cyl_skew,
    ""};

```

```

"ソフトセクタ方式      : ", s_soft_sec,
"ハードセクタ方式      : ", s_hard_sec,
"リム-ハーフ          : ", s_rmb,
"フォーマット         : ", s_surface,
""};

```

```

char    #ss_dform[] = {"SCSI-1Yn", "CCSYn", "SCSI-2Yn"};
uchar    s_unknown[] = {"不明Yn"};

char    #ss_yes_no[] = {"YESYn", "NOYn"};
char    #ss_surf[] = {"シリンダ順Yn", "ヘッド順Yn"};
char    s_graph[] = {"グラフフォーマット装置Yn"};

char    #ss_dtype[] = {
    /* 0 */ "磁気ディスクYn",
    /* 1 */ "磁気テープYn",
    /* 2 */ "ブリンクYn",
    /* 3 */ "ブロッコリーYn",
    /* 4 */ "単一書込多重読取装置Yn",
    /* 5 */ "CD-ROMYn",
    /* 6 */ "スキップYn",
    /* 7 */ "光磁気ディスクYn",
    /* 8 */ "磁気テープ/ディスクYn",
    /* 9 */ "通信装置Yn",
    /* A */ s_graph,
    /* B */ s_graph,
    ""};

char    s_logical[] = {
    "論理構成: シリンダ=%ld, ヘッド=%ld, セクタ/トラック=%ld, トラック=%ldYn"};

char    #ss_title[] = {
    "SU Version 1.1 (C) Copyright AUG 1993 by Y.MAKIYn",
    ""};

char    #help[] = {
    "起動例: A>SU 0Yn",
    ""};

char    #ss_help[] = {
    "----- コマンド・リファレンス -----Yn",
    "G - 緒元 リストYn",
    "I - Inquiry リストYn",
    "C<転送先 SCSI ID> - イメージングYn",
    "D[<開始ブロック>, <終了ブロック>] - セクタサイズYn",
    "E[<タイプ>, <ブロック>, <セクタ>] - セクタ内サイズYn",
    "S[<シーク回数>] - シークタイムYn",
    "R[<リード回数>] - リードタイムYn",
    "Q - ブロック終了Yn",
    "? - ヘルプYn",
    "----- 備考 -----Yn",
    "<タイプ>:= Byte, Word, Asciz(省略時 Byte)Yn",
    "<シーク回数>:= 省略時 1000(10進)Yn",
    "<リード回数>:= 省略時 512(10進)Yn",
    "<開始ブロック>, <終了ブロック>, <ブロック>, <セクタ>:= 10進で入力Yn",
    ""};

char    #id_error[] = {
    "SCSI-IDは 0-6 の範囲で入力してくださいYn",
    ""};

char    connection_error[] = {
    "SCSI-ID(%d)の装置がアクセスできませんYn"};

char    status_error[] = {
    "ID%d がGOOD(00)以外のステータス(%02x)を返してきましたYn"};

char    #ss_key[] = {
    "NO SENSE",
    "RECOVERED ERROR",
    "NOT READY",
    "MEDIUM ERROR",
    "HARDWARE ERROR",
    "ILLEGAL REQUEST",
    "UNIT ATTENTION",
    "DATA PROTECT",
    "BLANK CHECK",
    "Vendor Specific",
    "COPY ABORTED",
    "ABORTED COMMAND",
    "EQUAL",
    ""};

```



```

"VOLUME OVERFLOW",
"MISCOMPARE",
"RESERVED");
/* Extern Module & Data in ASPIASM.ASM */
extern void  *aspi_entry;
extern int   aspi_init();
extern int   aspi_exec(void *);
/*----- Utility */
int   geometry_exec(int);
int   inquiry_exec(int);
int   backup_exec(int, int);
int   seek_test(int, char *);
int   read_test(int, char *);
int   dump_exec(int, char *);
void   dump_byte(uchar *, long, int);
int   enter_exec(int, char *);
int   enter_byte(long, int);
int   enter_word(long, int);
int   enter_ascii(long, int);
int   make_byte_list(char *, uchar *);
int   make_word_list(char *, ushort *);
int   gethex(char *, int, char *);
int   ishex(char *);
int   isdec(char *);
int   touppers(char *);
int   get_skey(int, uchar);
void   char_disp(char **);
int   TokenSepa(char *, char **, int);
char   *TokenSrch(char *);
char   *TokenGet(char *, char *);
int   getstr(char *, int);
void   error_dsp(int);
ulong   bit_to_ulong(int, int, uchar *);
int   (*sesi_exec)(int, struct SCSI_PACKET *); /* 377 の発行 */
/*----- SCSI BIOS */
int   sesi_dataout(int, uchar *, int);
int   sesi_datain(int, uchar *, int);
int   sesi_messgin(int, uchar *, int);
int   sesi_stat(int, uchar *, int);
int   sesi_comout(int, uchar *, int);
int   sesi_select(int);
int   sesi_reset(void);
int   sesi_negate_ack(int);
int   nec_bios(int, struct SCSI_PACKET *);
int   aspi_bios(int, struct SCSI_PACKET *);
int   aspi_initialize(void);

void main(argc, argv)
int   argc;
char   *argv[];
{
char   key_buff[257];
int   kent, result;
int   i, s_id, d_id;
uchar   *p;
ulong   lng;
if((p = (uchar *)malloc(0x18000L)) == NULL){
printf("メモリが不足しています\n");
exit(1);
}
lng = (ulong)p;
/* PC-98vx 以前の DMA に対処 */
lng = (lng & 0xf0000000L) + 0x10000000L;
p = (uchar *)lng; /* for PC-98vx > */
readhd.buffer = p;
writehd.buffer = p;
if(argc != 2){
char_disp(help);
exit(1);
}
if(sscanf(argv[1], "%d", &s_id) != 1){
char_disp(help);
exit(1);
}
if((s_id > 6 || s_id < 0)){
char_disp(id_error);
exit(1);
}

```

```

for(i=0; i<17; i++){
s[i]=Token[i];

data_seg = FP_SEG(data_seg);

if(aspi_initialize() == TRUE)
sesi_exec = aspi_bios;
else
sesi_exec = nec_bios;

if(sesi_exec(s_id, &test_unit) == FAIL){
printf(connection_error, s_id);
exit(1);
}
inquiry_exec(s_id);
geometry_exec(s_id);
char_disp(ss_title);

do{
putch('~');
kent = getstr(key_buff, 79);
switch( toupper(key_buff[0]) ){
/* 0-9 */
case 'D':
result = dump_exec(s_id, key_buff+1);
break;
/* 10-19 */
case 'E':
result = enter_exec(s_id, key_buff+1);
break;
/* 20-29 */
case 'S':
result = seek_test(s_id, key_buff+1);
break;
/* 30-39 */
case 'R':
result = read_test(s_id, key_buff+1);
break;
/* Inquiry */
case 'I':
if(kent > 1)
result = ERROR;
else
char_disp(ss_inquiry);
break;
/* 40-49 */
case 'C':
i = TokenSepa(key_buff+1, s, 2);
if(i > 1){
result = ERROR;
break;
}
if(sscanf(s[0], "%d", &i) != 1){
result = ERROR;
break;
}
if((i == s_id) || (i > 6) || (i < 0)){
result = ERROR;
break;
}
d_id = i;
if(scsi_exec(d_id, &test_unit) == FAIL){
printf(connection_error, d_id);
break;
}
backup_exec(s_id, d_id);
break;
/* 50-59 */
case 'G':
if(kent > 1)
result = ERROR;
else{
char_disp(ss_geometry);
printf(s_logical, logic_cyl, heads,
logic_sec_trk, logic_blocks);
}
}
}

```



```

    }
    break;
/* 終了 */
case '?':
    if(kent > 1)
        result = ERROR;
    else
        char_disp(ss_help);
        result = NORMAL;
    }
    break;
/* 終了 */
case 'Q':
    if(kent > 1)
        result = ERROR;
    else result = NORMAL;
    break;
case 0:
    result = NORMAL;
    break;
default:
    result = ERROR;
    break;
}
if(result == ERROR){
    error_dsp(kent);
    result = NORMAL;
    continue;
}
}while(toupper(key_buff[0]) != 'Q');
}

/*
 * Inquiry - を得て表示する
 * id:SCSI-id
 */
int inquiry_exec(id)
int id;
{
    int i;
    if(scsi_exec(id,&inquiry) == FAIL){
        printf(connection_error, id);
        return(FAIL);
    }
    if(inquiry.status != STS_GOOD){
        get_skey(id, inquiry.status);
        return(STS_ERROR);
    }
    for(i = 3; i < 34; i += 2)
        strcpy(ss_inquiry[i], s_unknown);
    if(inquiry.data_length > 3){
        if((work[0] & 0x1f) <= 0x0b)
            strcpy(s_device_type, ss_dtype[work[0] & 0x1f]);
        device_type = work[0] & 0x1f;
        strcpy(s_rmb, ss_yes_no[(work[1] & 0x80) == 0]);
        sprintf(s_iso_ver, "%dYn", (work[2] & 0xc0) >> 6);
        sprintf(s_ecma_ver, "%dYn", (work[2] & 0x38) >> 3);
        sprintf(s_ansi_ver, "%dYn", work[2] & 0x07);
        strcpy(s_dform, ss_dform[work[3] & 0x0f]);
    }
    if(inquiry.data_length > 7){
        strcpy(s_reladr, ss_yes_no[(work[7] & 0x80) == 0]);
        strcpy(s_wbus32, ss_yes_no[(work[7] & 0x40) == 0]);
        strcpy(s_wbus16, ss_yes_no[(work[7] & 0x20) == 0]);
        strcpy(s_sync, ss_yes_no[(work[7] & 0x10) == 0]);
        strcpy(s_linked, ss_yes_no[(work[7] & 0x08) == 0]);
        strcpy(s_cmdque, ss_yes_no[(work[7] & 0x02) == 0]);
        strcpy(s_sftre, ss_yes_no[(work[7] & 0x01) == 0]);
        memmove(s_vend_id, work+8, 8);
        s_vend_id[8] = 0;
        i = strlen(s_vend_id);
        s_vend_id[i] = LF;
        s_vend_id[i+1] = 0;
        memmove(s_prod_id, work+16, 16);
        s_prod_id[16] = 0;
        i = strlen(s_prod_id);
        s_prod_id[i] = LF;
    }
}

```

```

    s_prod_id[i+1] = 0;
    memmove(s_prod_rev, work+32, 4);
    s_prod_rev[4] = 0;
    i = strlen(s_prod_rev);
    s_prod_rev[i] = LF;
    s_prod_rev[i+1] = 0;
}
return(NORMAL);
}

/*
 * Read Capacity, Mode Sense(PAGE 3, 4)から
 * 諸元 - を得る
 * id:SCSI-id
 */
int geometry_exec(id)
int id;
{
    int i, j;
    uchar *p;
    uchar pg_code, page[2][48];
    unsigned ui, uj, pblock_len, track_zone;
    unsigned alsec_zone, altrk_zone, altrk_lunit, sec_track;
    ulong v1, v2, cylinders, sec_cyl;
    for(i = 3; i < 38; i += 2)
        strcpy(ss_geometry[i], s_unknown);
/*
 * READ CAPACITY COMMAND
 */
    if(scsi_exec(id,&read_capa) == FAIL){
        printf(connection_error, id);
        return(FAIL);
    }
    if(read_capa.status != STS_GOOD){
        get_skey(id, read_capa.status);
        return(STS_ERROR);
    }
    /* END BLOCK 71Vxなので +1 する */
    lblock_total = bit_to_ulong(7, 32, read_capa.buffer) + 1;
    sprintf(s_lblock_total, "%ldYn", lblock_total);
    v2 = (unsigned)bit_to_ulong(7, 32, read_capa.buffer+4);
    lblock_len = (unsigned)v2;
    sprintf(s_lblock_len, "%ld n {Yn", v2);
    v1 = lblock_total * v2 / 1000;
    sprintf(s_capacity, "%ld 4n {Yn", v1);

    for(i = 0; i < 2; i++)
        for(j = 0; j < 48; j++)
            page[i][j] = 0;
/*
 * MODE SENSE(page3, 4)で 3Vx - を得る
 */
    for(pg_code = 3; pg_code < 5; pg_code++){
        *pgcode_pointer = pg_code; /* 3 or 4 */
        memset(work, sizeof(work), 0);
        if(scsi_exec(id,&mode_sense) == FAIL){
            printf(connection_error, id);
            return(FAIL);
        }
        if(mode_sense.status != STS_GOOD){
            get_skey(id, mode_sense.status);
            return(FAIL);
        }
        if(mode_sense.data_length > 0){
            p = mode_sense.buffer;
            p += 3; /* +block descriptor length */
            i = (int)*p;
            p += i;
            p++;
            *p &= 7;
            while(*p != 0){
                p++;
                i = (int)*p;
                p++;
                j = 0;
                while(i){

```



```

        page[pg_code - 3][j] = *p++;
        i--;
        j++;
    }
}
/*
 * PAGE3 の作成
 */
track_zone = (unsigned)bit_to_ulong(7, 16, page[0]);
sprintf(s_track_zone, "%dYn", track_zone);
alsec_zone = (unsigned)bit_to_ulong(7, 16, page[0]+2);
sprintf(s_alsec_zone, "%dYn", alsec_zone);
altrk_zone = (unsigned)bit_to_ulong(7, 16, page[0]+4);
sprintf(s_altrk_zone, "%dYn", altrk_zone);
altrk_lunit = (unsigned)bit_to_ulong(7, 16, page[0]+6);
sprintf(s_altrk_lunit, "%dYn", altrk_lunit);
sec_track = (unsigned)bit_to_ulong(7, 16, page[0]+8);
sprintf(s_sec_track, "%dYn", sec_track);
pblock_len = (unsigned)bit_to_ulong(7, 16, page[0]+10);
sprintf(s_pblock_len, "%d N' %dYn", pblock_len);
ui = (unsigned)bit_to_ulong(7, 16, page[0]+12);
sprintf(s_interleave, "%dYn", ui);
ui = (unsigned)bit_to_ulong(7, 16, page[0]+14);
sprintf(s_trk_skew, "%dYn", ui);
ui = (unsigned)bit_to_ulong(7, 16, page[0]+16);
sprintf(s_cyl_skew, "%dYn", ui);
strcpy(s_soft_sec, ss_yes_no[(page[0][18] & 0x80) == 0]);
strcpy(s_hard_sec, ss_yes_no[(page[0][18] & 0x40) == 0]);
strcpy(s_rmb, ss_yes_no[(page[0][18] & 0x20) == 0]);
strcpy(s_surface, ss_surf[(page[0][18] & 0x10) != 0]);
/*
 * PAGE4 の作成
 */
cylinders = bit_to_ulong(7, 24, &page[1][0]);
sprintf(s_cylinder, "%dYn", cylinders);
heads = (unsigned)page[1][3];
sprintf(s_heads, "%dYn", heads);
/*
 * 論理的な構成の作成
 */
if(pblock_len == 0) /* 物理セクタ長 0 に対応 */
    pblock_len = lblock_len;
if(lblock_len > pblock_len) {
    ui = lblock_len / pblock_len;
    alsec_zone /= ui;
    altrk_zone /= ui;
    sec_track /= ui;
} else if(lblock_len < pblock_len) {
    ui = pblock_len / lblock_len;
    alsec_zone *= ui;
    altrk_zone *= ui;
    sec_track *= ui;
}
vl = cylinders * heads * sec_track;
if(vl == lblock_total) /* 交代を含まず記述しているか? */
    logic_cyl = cylinders;
    logic_sec_trk = sec_track;
    logic_blocks = lblock_total;
    return(NORMAL);
}
if(track_zone > 0) {
    /* 有効セクタ/トラックを求める */
    ui = track_zone - altrk_zone;
    logic_sec_trk = (ui * sec_track - alsec_zone) / ui;
    if(altrk_zone > 0) {
        if(track_zone <= heads)
            heads = heads - altrk_zone;
        else
            heads = track_zone - altrk_zone;
    }
} else {
    if(alsec_zone < (sec_track / 4)) {

```

```

        /* 古いサイズの SCSI でトラッキング中の交代セクタがあるか?
        */
        ui = sec_track - alsec_zone;
        vl = heads * ui;
        if(lblock_total % vl)
            logic_sec_trk = sec_track;
        else
            logic_sec_trk = ui;
        else logic_sec_trk = sec_track;
    }
    sec_cyl = logic_sec_trk * heads;
    logic_cyl = lblock_total / sec_cyl;
    logic_blocks = logic_cyl * heads * logic_sec_trk;
    return(NORMAL);
}
/*
 * 7'0で複写
 * s_id:送り側SCSI-ID
 * d_id:受け側SCSI-ID
 */
int backup_exec(s_id, d_id)
int s_id;
int d_id;
{
    int result, c_id;
    long i, cnt;
    unsigned sbk_len, dbk_len, sbk_unit, dbk_unit, mod;
    ulong sbk_total, dbk_total, s_capa, d_capa, xfer_capa;
    ulong xblk_total, s_lba, d_lba;
    if(scsi_exec(s_id, &read_capa) == FAIL) {
        printf(connection_error, s_id);
        return(FAIL);
    }
    if(read_capa.status != STS_GOOD) {
        get_skey(s_id, read_capa.status);
        return(STS_ERROR);
    }
    sbk_total = bit_to_ulong(7, 32, read_capa.buffer) + 1;
    sbk_len = (unsigned)bit_to_ulong(7, 32, read_capa.buffer+4);
    s_capa = sbk_total * sbk_len;
    if(scsi_exec(d_id, &read_capa) == FAIL) {
        printf(connection_error, d_id);
        return(FAIL);
    }
    if(read_capa.status != STS_GOOD) {
        get_skey(d_id, read_capa.status);
        return(STS_ERROR);
    }
    dbk_total = bit_to_ulong(7, 32, read_capa.buffer) + 1;
    dbk_len = (unsigned)bit_to_ulong(7, 32, read_capa.buffer+4);
    d_capa = dbk_total * dbk_len;
    if(s_capa <= d_capa) {
        xfer_capa = s_capa;
        xblk_total = sbk_total;
        c_id = s_id;
    } else {
        xfer_capa = d_capa;
        xblk_total = dbk_total;
        c_id = d_id;
    }
    sbk_unit = BUFF_SIZE / sbk_len;
    dbk_unit = BUFF_SIZE / dbk_len;
    readhd.data_length = BUFF_SIZE;
    writehd.data_length = BUFF_SIZE;
    cnt = xfer_capa / BUFF_SIZE;
    mod = xfer_capa % BUFF_SIZE;
    printf("転送7'0サイズ=%dYn", xblk_total);
    printf("ID %d(%d:%d) から ID %d(%d:%d) へ 転送(Y/N)? ",
        s_id, sbk_total, sbk_len, d_id, dbk_total, dbk_len);
    while(TRUE) {
        result = toupper(getch());
        if(result == 'Y') {
            putch(result);
            printf("Yn");
            break;
        } else if(result == 'N') {

```



```

        putch(result);
        return(NORMAL);
    }
}
for(i = s_lba = d_lba = 0L; i < cnt; i++){
    if(kbhit())
        if(getch() == 0x1b)
            return(NORMAL);
    if((result = read_out(s_id, s_lba, sbk_unit)) != NORMAL)
        return(result);
    if((result = write_out(d_id, d_lba, dbk_unit)) != NORMAL)
        return(result);
    putch(CR);
    s_lba += sbk_unit;
    d_lba += dbk_unit;
    if(s_id == c_id)
        printf("%ld", s_lba);
    else printf("%ld", d_lba);
}
if(mod > 0){
    sbk_unit = mod / sbk_len;
    dbk_unit = mod / dbk_len;
    readhd.data_length = mod;
    writehd.data_length = mod;
    if((result = read_out(s_id, s_lba, sbk_unit)) != NORMAL)
        return(result);
    if((result = write_out(d_id, d_lba, dbk_unit)) != NORMAL)
        return(result);
    putch(CR);
    if(s_id == c_id)
        printf("%ld", s_lba + sbk_unit);
    else printf("%ld", d_lba + dbk_unit);
}
printf("\n転送が終了しました\n");
return(NORMAL);
}

int seek_test(id, p)
int id;
char *p;
{
    ulong cyl, tmp, lba[2];
    time_t s_sec, e_sec, o_sec, sec; /* typedef long */
    int i, seek_count, result;
    seek_count = SEEK_COUNT;
    i = TokenSeps(p, s, 2);
    if(i > 1)
        return(ERROR);
    if(i == 1)
        if(sscanf(Token[0], "%d", &seek_count) != 1)
            return(ERROR);
    /*
     * シリタの3分の1のワツヲヲヲヲを求める
     */
    if(device_type == 0){ /* ハードディスク */
        cyl = logic_cyl / 3;
        tmp = cyl * heads * logic_sec_trk;
    } else if(device_type == 7){ /* 3.5インチFDDかも */
        cyl = lblock_total / 25 / 3;
        tmp = cyl * 25;
    } else return(NORMAL);
    printf("シリタ 0-%ld(LBA:%ld) 間で%d回ワツ(Y/N)?", cyl, tmp, seek_count);
    for(;;){
        i = toupper(getch());
        if(i == 'Y'){
            putch(i);
            printf("\n");
            break;
        } else if(i == 'N'){
            putch(i);
            printf("\n");
            return(NORMAL);
        }
    }
    lba[0] = lba[1] = 0L;
    /*
     * ハードの秒数を調べる

```

```

    */
    /* ハードを定位置に固定 */
    if((result = seek_out(id, 0L)) != NORMAL)
        return(result);
    printf("ハードのワツ開始\n");
    time(&s_sec);
    for(i = 0; i < seek_count; i++){
        if((result = seek_out(id, lba[i&1])) != NORMAL)
            return(result);
        if(seekhd.status != STS_GOOD)
            return(NORMAL);
        if(kbhit())
            if(getch() == ESC)
                return(NORMAL);
    }
    time(&e_sec);
    o_sec = e_sec - s_sec;
    printf("ハードのワツ秒数:%ld\n", o_sec);

    lba[1] = tmp;
    /*
     * ワツの秒数を調べる
     */
    /* ハードを定位置に固定 */
    if((result = seek_out(id, 0L)) != NORMAL)
        return(result);
    printf("ワツ開始\n");
    time(&s_sec);
    for(i = 0; i < seek_count; i++){
        if((result = seek_out(id, lba[i&1])) != NORMAL)
            return(result);
        if(seekhd.status != STS_GOOD)
            return(NORMAL);
        if(kbhit())
            if(getch() == ESC)
                return(NORMAL);
    }
    time(&e_sec);
    sec = e_sec - s_sec;
    printf("ワツ秒数:%ld\n", sec);
    printf("ワツ秒数からハードの秒数を引いた秒数:%ld\n", sec - o_sec);
    printf("平均ワツタイム:%ldms\n", (sec - o_sec) * 1000 / seek_count);
    return(NORMAL);
}

int read_test(id, p)
int id;
char *p;
{
    time_t s_sec, e_sec, o_sec, sec; /* typedef long */
    unsigned lba_unit;
    ulong max_lba, lba;
    int i, read_count, result, tmp;

    lba_unit = BUFF_SIZE / lblock_len;
    max_lba = lblock_total - lba_unit;
    readhd.data_length = BUFF_SIZE;
    read_count = READ_COUNT;
    i = TokenSeps(p, s, 2);
    if(i > 1)
        return(ERROR);
    if(i == 1)
        if(sscanf(Token[0], "%d", &read_count) != 1)
            return(ERROR);
        if(((long)read_count * BUFF_SIZE) < 0x100000L){
            printf("ワツワツを%d以上にしてください\n",
                0x100000L / BUFF_SIZE);
            return(NORMAL);
        }
    }
    printf("ワツワツハードのワツ(Y/N)?",
        (ulong)read_count * BUFF_SIZE / 0x100000L);
    for(;;){
        i = toupper(getch());
        if(i == 'Y'){
            putch(i);
            printf("\n");
            break;

```



```

    } else if (i == 'N') {
        putchar(i);
        printf("Yn");
        return(NORMAL);
    }
}
/*
 * 1-1-1 の秒数を調べる
 */
/*
 * ヘッドを定位置に固定
 */
if ((result = seek_out(id, 0L)) != NORMAL)
    return(result);
printf("1-1-1 のシーク開始Yn");
time(&s_sec);
for (i = 0, lba = 0L; i < read_count; i++) {
    if ((result = seek_out(id, lba)) != NORMAL)
        return(result);
    if (seekhd.status != STS_GOOD)
        return(NORMAL);
    if (kbhit())
        if (getch() == ESC)
            return(NORMAL);
    lba += lba_unit;
    if (lba > max_lba)
        lba = 0L;
}
time(&e_sec);
o_sec = e_sec - s_sec;
printf("1-1-1 のシーク秒数: %ldYn", o_sec);
/*
 * ドライブのキャパシティを満たしておく (2MB 以上)
 * ためのシーク
 */
tmp = (int)(0x200000L / BUFF_SIZE);
printf("キャパシティへの読み込み (2MB 以上) 開始Yn");
for (i = 0, lba = 0L; i < tmp; i++, lba += lba_unit) {
    if ((result = read_out(id, lba, lba_unit)) != NORMAL)
        return(result);
    if (readhd.status != STS_GOOD)
        return(NORMAL);
}
/*
 * リードの秒数を調べる
 */
printf("リード開始Yn");
time(&s_sec);
for (i = 0; i < read_count; i++) {
    if ((result = read_out(id, lba, lba_unit)) != NORMAL)
        return(result);
    if (readhd.status != STS_GOOD)
        return(NORMAL);
    if (kbhit())
        if (getch() == ESC)
            return(NORMAL);
    lba += lba_unit;
    if (lba > max_lba)
        lba = 0L;
}
time(&e_sec);
sec = e_sec - s_sec;
if (sec == 0) {
    printf("リード秒数が少なすぎますYn");
    return(NORMAL);
}
printf("リード秒数: %ldYn", sec);
printf("リード秒数から1-1-1の秒数を引いた秒数: %ldYn", sec - o_sec);
printf("リード転送レート: %ldKB/s (1024)Yn",
        (long)read_count * BUFF_SIZE / sec / 1024);
return(NORMAL);
}

int dump_exec(id, str)
int id;
char *str;
{
    int i, j, c;

```

```

    long from, to, lcnt;
    char *p;
    uchar *w;
    p = str;
    touppers(p);
    i = TokenSeps(p, s, 3);
    if (i > 2)
        return(ERROR);
    if (i == 0) { /* not address & range */
        from = crr_block;
        if (from > (lblock_total-1))
            from = crr_block = 0L;
        to = from + 1L;
        crr_block = to;
    } else if (i == 1) { /* address(range) */
        if (isdec(Token[0]) == NO)
            return(ERROR);
        else {
            if (sscanf(Token[0], "%ld", &from) != 1)
                return(ERROR);
            if (from > (lblock_total-1))
                return(ERROR);
        }
        if (i == 2) {
            if (isdec(Token[1]) == NO)
                return(ERROR);
            else {
                if (sscanf(Token[1], "%ld", &to) != 1)
                    return(ERROR);
                if (from > to)
                    return(ERROR);
                if (to > (lblock_total-1))
                    return(ERROR);
                to++;
                crr_block = to;
            }
        } else { /* not range */
            to = from + 1;
            crr_block = to;
        }
    }
    readhd.data_length = lblock_len;
    j = lblock_len / 16;
    while (from < to) {
        if (read_out(id, from, 1) != NORMAL)
            break;
        if (kbhit()) {
            c = getch();
            if (c == ESC) break;
        }
        printf("0x%04X ~ 0x%04X = %ldYn", from);
        for (i = 0, lcmt = 0L, w = readhd.buffer;
             i < j; i++, w += 16, lcmt += 16L)
            dump_byte(w, lcmt, 16);
        from++;
    }
    return(NORMAL);
}

void dump_byte(p, adrs, cnt)
uchar *p;
long adrs;
int cnt; /* max 16bytes */
{
    int i, j;
    char *sp, s[76];
    sprintf(s, "%04X", adrs);
    memset(s+5, ' ', 65);
    sprintf(s+70, "Yn");
    for (sp = s+5, i = 0, j = 54; i < cnt; sp += 3, i++, j++) {
        sprintf(sp, "%02X", *p);
        if ((*p >= 0x20) && (*p <= 0x7e))
            s[j] = *p++;
        else s[j] = ' ';
        p++;
    }
}

```



```

        if(i == 7)
            *(sp+2) = '-';
    }
    *sp = '-';
    printf("%s", s);
}

int  enter_exec(id, str)
int  id;
char *str;
{
    int  i, c, length, type, off;
    long from;
    char *p;
    char s_yn[] = {"? 0+? (%id) の内容を変更しますか?(Y/N)"};
    uchar wk[80];
    p = str;
    switch(*p){
        case 'A':case 'a': /* Ascii */
            type = 2;
            p++;
            break;
        case 'B':case 'b': /* Byte */
            type = 0;
            p++;
            break;
        case 'W':case 'w': /* Word */
            type = 1;
            p++;
            break;
        default:
            type = 0;
            break;
    }
    i = TokenSepa(p, s, 4);
    if(i < 2)
        return(ERROR);
    if(isdec(Token[0]) == NO)
        return(ERROR);
    else{
        if(sscanf(Token[0], "%ld", &from) != 1)
            return(ERROR);
        if(from > lblock_total)
            return(ERROR);
    }
    if(isdec(Token[1]) == NO)
        return(ERROR);
    else{
        if(sscanf(Token[1], "%d", &off) != 1)
            return(ERROR);
        if(off >= lblock_len)
            return(ERROR);
    }
    p = TokenSrch(p);
    p = TokenGet(p, wk);
    p = TokenSrch(p);
    p = TokenGet(p, wk);
    readhd.data_length = lblock_len;
    writehd.data_length = lblock_len;
    if(read_out(id, from, 1) != NORMAL)
        return(NORMAL);
    if(i > 2){
        if(type == 0){ /* byte */
            if((length = make_byte_list(p, wk)) == ERROR)
                return(ERROR);
        }
        else if(type == 1){ /* word */
            if((length = make_word_list(p, (ushort *)wk)) == ERROR)
                return(ERROR);
            length *= 2;
        }
        else if(type == 2){ /* ascii */
            p++;
            length = strlen(p);
            memcpy(wk, p, length);
        }
        memcpy(readhd.buffer+off, wk, length);
        printf(s_yn, from);
        c = getch();
    }
}

```

```

        putchar(c);
        putchar(CR);
        putchar(LF);
        if(toupper(c) == 'Y')
            write_out(id, from, 1);
    }
    else{ /* non list mode */
        if(type == 0){ /* byte */
            if(enter_byte(from, off) == ERROR)
                return(ERROR);
        }
        else if(type == 1){ /* word */
            if((off + 2L) > lblock_len)
                return(ERROR);
            if(enter_word(from, off) == ERROR)
                return(ERROR);
        }
        else if(type == 2){ /* ascii */
            if(enter_ascii(from, off) == ERROR)
                return(ERROR);
        }
    }
    printf(s_yn, from);
    c = getch();
    putchar(c);
    putchar(CR);
    putchar(LF);
    if(toupper(c) == 'Y')
        write_out(id, from, 1);
}
return(NORMAL);
}

int  enter_byte(block, adrs)
long block;
int  adrs;
{
    char direct, c, str[16], key_buff[67];
    uchar wk[80];
    int  kent, length;
    while(TRUE){
        wk[0] = readhd.buffer[adrs];
        c = wk[0];
        if((c < ' ') || (c > '~'))
            c = '.';
        sprintf(str, "%07ld:%04d(%03X) %02X%c",
            block, adrs, adrs, wk[0], c);
        printf(str);
        kent = gethex(key_buff, sizeof(key_buff) - 1, &direct);
        if(kent == 0){
            if(direct == '-'){
                adrs++;
                if(adrs >= lblock_len)
                    adrs = 0L;
            }
            else if(direct == '+'){
                adrs--;
                if(adrs < 0)
                    adrs = lblock_len - 1;
            }
            else
                break; /* CR */
        }
        else{
            if((length = make_byte_list(key_buff, wk)) == ERROR)
                return(ERROR);
            if(adrs + length > lblock_len)
                return(ERROR);
            memcpy(readhd.buffer+adrs, wk, length);
            if(direct == '-'){
                adrs--;
                if(adrs < 0)
                    adrs = lblock_len - 1;
            }
            else{
                adrs += length;
                if(adrs >= lblock_len)
                    adrs = 0;
            }
        }
    }
}
return(NORMAL);

```



```

    }

    int  enter_word(block, adrs)
    long block;
    int  adrs;
    {
    char  direct, str[16], key_buff[78];
    short kent, length;
    ushort wk[80];
    while(TRUE) {
        wk[0] = *((unsigned short *) (readhd.buffer+adrs));
        sprintf(str, "%07ld:%04d[%03X] %04X.",
            block, adrs, adrs, wk[0]);
        printf(str);
        kent = gethex(key_buff, sizeof(key_buff) - 1, &direct);
        if(kent == 0) {
            if(direct == '+') {
                adrs += 2;
                if(adrs == (lblock_len + 1))
                    break;
                if(adrs >= lblock_len)
                    adrs = 0;
            }
            else if(direct == '-') {
                adrs -= 2;
                if(adrs == -1)
                    break;
                if(adrs < 0)
                    adrs = lblock_len - 2;
            }
            else break; /* CR */
        }
        else {
            if((length = make_word_list(key_buff, wk)) == ERROR)
                return(ERROR);
            memcpy(readhd.buffer+adrs, (unsigned char *) wk, 2);
            if(direct == '-') {
                adrs -= 2;
                if(adrs == -1)
                    break;
                if(adrs < 0L)
                    adrs = lblock_len - 2;
            }
            else {
                adrs += 2;
                if(adrs == (lblock_len + 1))
                    break;
                if(adrs >= lblock_len)
                    adrs = 0;
            }
        }
    }
    return(NORMAL);
}

```

```

int  enter_ascii(block, adrs)
long block;
int  adrs;
{
    char  c, str[16], key_buff[67];
    uchar wk[80];
    int  kent;
    while(TRUE) {
        wk[0] = readhd.buffer[adrs];
        c = wk[0];
        if((c < ' ') || (c > '~'))
            c = '?';
        sprintf(str, "%07ld:%04d[%03X] '%c' %02X.",
            block, adrs, adrs, c, wk[0]);
        printf(str);
        kent = getstr(key_buff, 1);
        if(kent == 0)
            break;
        else {
            wk[0] = key_buff[0];
            readhd.buffer[adrs] = wk[0];
            adrs++;
            if(adrs >= lblock_len)
                adrs = 0L;
        }
    }
}

```

```

    }
    return(NORMAL);
}

int  make_byte_list(s, d)
char  *s;
uchar *d;
{
    int  i, cnt;
    ushort tmp;
    char wk[80];
    i = cnt = 0;
    while(*s != 0) {
        s = TokenSrch(s);
        if(*s == 0)
            return(cnt);
        else s = TokenGet(s, wk);
        if(ishex(wk) == NO)
            return(ERROR);
        if((i = strlen(wk)) > 2)
            i -= 2;
        else i = 0;
        sscanf(wk+i, "%X", &tmp);
        *d++ = tmp;
        cnt++;
    }
    return(cnt);
}

```

```

int  make_word_list(s, d)
char  *s;
ushort *d;
{
    int  i, cnt;
    char wk[80];
    i = cnt = 0;
    while(*s != 0) {
        s = TokenSrch(s);
        if(*s == 0)
            return(cnt);
        else s = TokenGet(s, wk);
        if(ishex(wk) == NO)
            return(ERROR);
        if((i = strlen(wk)) > 4)
            i -= 4;
        else i = 0;
        sscanf(wk+i, "%X", d++);
        cnt++;
    }
    return(cnt);
}

```

```

int  gethex(s, max_char, end_char)
char  *s;
int  max_char;
char  *end_char;
{
    char  *p, c;
    int  char_cnt;
    p = s;
    char_cnt = 0;
    do {
        c = getch();
        c = toupper(c);
        if(c == ESC) {
            while(getch() == ESC)
                continue;
        }
        else if((c == 0x08) && (p > s)) /* BS */
            putchar(c);
        putchar(c);
        p++;
        char_cnt++;
    }
    else if(((c >= '0' && c <= '9') || (c >= 'A' && c <= 'F')) &&

```



```

        (max_char > char_cnt)){
            *p++ = c;
            char_cnt++;

            putchar(c);
        }else if(c == '\n'){
            putchar(c);
            break;
        }else if((c == ' ')&&(char_cnt == 0))
            break;
    }while(c != CR);
    *end_char = c;
    putchar(CR);
    putchar(LF);
    *p = 0;
    return(char_cnt);
}

```

```

int ishex(s)
char *s;
{
    while(*s != 0)
        if(isxdigit(*s++) == NO)
            return(NO);
    return(YES);
}

```

```

int isdec(s)
char *s;
{
    while(*s != 0)
        if(isdigit(*s++) == NO)
            return(NO);
    return(YES);
}

```

```

int touppers(p)
char *p;
{
    int i;
    for(i = 0; *p != 0; p++, i++)
        *p = toupper(*p);
    return(i);
}

```

```

/*
 * Read(6n bit)コマンドの発行
 * id:SCSI-ID
 * lba:論理アドレス
 * xlen:転送バイト数
 */
int read_out(id, lba, xlen)
int id;
ulong lba;
unsigned xlen;
{
    *read_blk_pointer = (uchar)xlen;
    *(read_lba_pointer+2) = (uchar)lba;
    lba >>= 8;
    *(read_lba_pointer+1) = (uchar)lba;
    lba >>= 8;
    *(read_lba_pointer+0) = (uchar)lba;
    if(scsi_exec(id, &readhd) == FAIL){
        printf(connection_error, id);
        return(FAIL);
    }
    if(readhd.status != STS_GOOD){
        get_skey(id, readhd.status);
        return(STS_ERROR);
    }
    else return(NORMAL);
}

```

```

/*
 * Write(6n bit)コマンドの発行
 * id:SCSI-ID
 * lba:論理アドレス
 * xlen:転送バイト数
 */

```

```

int write_out(id, lba, xlen)
int id;
ulong lba;
unsigned xlen;
{
    *write_blk_pointer = (uchar)xlen;
    *(write_lba_pointer+2) = (uchar)lba;
    lba >>= 8;
    *(write_lba_pointer+1) = (uchar)lba;
    lba >>= 8;
    *(write_lba_pointer+0) = (uchar)lba;
    if(scsi_exec(id, &writehd) == FAIL){
        printf(connection_error, id);
        return(FAIL);
    }
    if(writehd.status != STS_GOOD){
        get_skey(id, writehd.status);
        return(STS_ERROR);
    }
    return(NORMAL);
}

```

```

/*
 * Seek(6n bit)コマンドの発行
 * id:SCSI-ID
 * lba:論理アドレス
 */

```

```

int seek_out(id, lba)
int id;
ulong lba;
{
    *(seek_lba_pointer+2) = (uchar)lba;
    lba >>= 8;
    *(seek_lba_pointer+1) = (uchar)lba;
    lba >>= 8;
    *(seek_lba_pointer+0) = (uchar)lba;
    if(scsi_exec(id, &seekhd) == FAIL){
        printf(connection_error, id);
        return(FAIL);
    }
    if(seekhd.status != STS_GOOD){
        printf(status_error, seekhd.status);
        return(STS_ERROR);
    }
    else return(NORMAL);
}

```

```

/*
 * もしSCSIエラーが CHECK CONDITION であれば
 * Request senseコマンドを発行して Sense Key を返す
 * id:SCSI-ID
 * status:SCSIエラー
 */

```

```

int get_skey(id, status)
int id;
uchar status;
{
    char str[32];
    if(status != STS_CHK_CONDITION){
        printf(status_error, id, status);
        return(FALSE);
    }
    printf("ID %d が %s のエラーを返してきました\n",
        id, ss_skey[sense_buffer[2] & 0x0f]);
    return(sense_buffer[2] & 0x0f);
}

```

```

/*
 * バイト配列をMSB, LSB逆にしてLONG変数に格納
 * bit_ptr:ビットポインタ
 * bit_len:ビット長
 * p:6n bitポインタ
 */

```

```

ulong bit_to_ulong(bit_ptr, bit_len, p)
int bit_ptr;
int bit_len;

```



```

uchar *p;
{
int cry;
uchar adc;
ulong val, rig;
cry = 7 - bit_ptr;
adc = 0x80;
adc >>= cry;
val = 0L;
rig = 0x00000001L;
rig <<= (bit_len - 1);
do{
if((p & adc) != 0)
val |= rig;
cry++;
adc >>= 1;
rig >>= 1;
if(cry > 7){
cry = 0;
p++;
adc = 0x80;
}
}while(--bit_len);
return(val);
}

/*
 * 文字列の表示
 * p: 文字配列の* 1つ(文字列で終了)
 */
void char_disp(p)
char **p;
{
int i;
i = 0;
while(p[i][0] != (char)NULL)
printf(p[i++]);
}

/*
 * 文字列を字句に分解
 */
static int TokenSeps(char *s, char **d, int max)
{
int i;
for(i = 0; i < max; i++){
s = TokenSrch(s);
if(*s == 0)
return(i);
else if(*s == '\n')
return(i);
else s = TokenGet(s, *d++);
}
return(i);
}

/*
 * 字句の始めを見つける
 */
static char *TokenSrch(char *p)
{
int i;
for(i = 0; i < 79; i++, p++){
if(*p == ' ')
continue;
if(*p == '\t')
continue;
if(*p == ','){
if(*(p-1) == ',')
break;
else continue;
}
break;
}
return(p);
}

```

```

}

/*
 * 字句の取り出し
 */
static char *TokenGet(char *s, char *d)
{
int len;
char *p;
if(*s == '"')
if((p = strchr(++s, '"')) != (char *)NULL){
*p++ = 0;
strcpy(d, s);
return(p);
}
for(;;){
if(*s == 0) break;
if(*s == '\n') break;
if(*s == ',') break;
if(*s == '\t') break;
if(*s == '\r') break;
*d++ = *s++;
}
*d = 0;
return(s);
}

/*
 * 文字列の入力
 */
int getstr(s, max_char)
char *s;
int max_char;
{
char *p, c;
int char_cnt;
p = s;
char_cnt = 0;
do{
c = getch();
if(c == ESC){
while(getch() == ESC)
continue;
}else if(c == 0x08){
if(p > s){
putch(c);
putch(0x20);
putch(c);
p--;
char_cnt--;
}
}else if((c >= 0x20) && (c <= 0x7e) && (max_char > char_cnt)){
*p++ = c;
char_cnt++;
putch(c);
}
}while(c != CR);
putch(CR);
putch(LF);
*p = 0;
return(char_cnt);
}

void error_dsp(length)
int length;
{
do
putch(0x20);
while((--length) && (length < 70));
printf("\n Error\n");
}

/*
 * NEC SCSI BIOS
 */

```



```
int nec_bios(int dn, struct SCIPACKET *ctrl)
{
    int comp;

    if(scsi_select(dn) != 0)
        return(FAIL);
    comp = 0;
    while(comp == 0) {
        switch(rsl.h.ah) {
            case 0x2a:
            case 0x3a:
            case 0x0a:
                if(scsi_comout(dn, ctrl->cdb, ctrl->cdb_length) != 0)
                    goto errquit;
                break;
            case 0x28:
            case 0x38:
            case 0x08:
                if(scsi_dataout(dn, ctrl->buffer, ctrl->data_length) != 0)
                    goto errquit;
                break;
            case 0x29:
            case 0x39:
            case 0x09:
                if(scsi_datain(dn, ctrl->buffer, ctrl->data_length) != 0)
                    goto errquit;
                break;
            case 0x2b:
            case 0x3b:
            case 0x0b:
                if(scsi_stat(dn, &ctrl->status, 1) != 0)
                    goto errquit;
                break;
            case 0x2f:
            case 0x3f:
            case 0x0f:
                if(scsi_messgin(dn, &ctrl->message, 1) != 0)
                    goto errquit;
                break;
            case 0x10:
                if(scsi_negate_ack(dn) != 0)
                    goto errquit;
                comp = -1;
                break;
            case 0x21:
                comp = -1;
                break;
            default:
                errquit:
                printf("継続処理不能なエラーが発生しました\n");
                return(FAIL);
        }
    }
    if(ctrl->status == STS_CHK_CONDITION)
        scsi_exec(dn, &request_sense);
    return(NORMAL);
}
```

```
int scsi_reset(void)
{
    cpu.h.ah = 0;
    cpu.h.al = 0xc0;
    cpu.x.dx = 0x0080;
    seg.es = seg.ds = data_seg;
    int86x(SCSI BIOS, &cpu, &rsl, &seg);
    return(rsl.x.cflag);
}

int scsi_select(int dn)
{
    cpu.h.ah = 7;
    cpu.h.al = 0xc0 + (dn & 0xf);
    cpu.x.dx = 0x0080;
    seg.es = seg.ds = data_seg;
    int86x(SCSI BIOS, &cpu, &rsl, &seg);
    return(rsl.x.cflag);
}

int scsi_comout(int dn, uchar *buf, int siz)
{
    cpu.h.ah = 0x1a;
    cpu.h.al = 0xc0 + (dn & 0xf);
    cpu.x.dx = 0x0080;
    switch(*buf >> 5) {
        case 0:
            cpu.x.cx = 6;
            break;
        case 1:
            cpu.x.cx = 10;
            break;
        case 5:
            cpu.x.cx = 12;
            break;
        default:
            cpu.x.cx = siz;
    }
    cpu.x.bx = FP_OFF(buf);
    seg.es = FP_SEG(buf);
    seg.ds = data_seg;
    int86x(SCSI BIOS, &cpu, &rsl, &seg);
    return(rsl.x.cflag);
}

int scsi_stat(int dn, uchar *buf, int siz)
{
    cpu.h.ah = 0x1b;
    cpu.h.al = 0xc0 + (dn & 0xf);
    cpu.x.dx = 0x0080;
    cpu.x.cx = siz;
    cpu.x.bx = FP_OFF(buf);
    seg.es = FP_SEG(buf);
    seg.ds = data_seg;
    int86x(SCSI BIOS, &cpu, &rsl, &seg);
    return(rsl.x.cflag);
}
```

■ ディスク頒布について ■

本稿で紹介した SCSI ユーティリティ「SU」のディスク頒布を行います。

機能は本稿で紹介したとおりで、

SU.C

ASPIASM.ASM

SU.EXE

からなります。

基本動作環境は PC-9801/MS-DOS です。

●頒布価格は 3,000 円(送料・税込み)です。

ご希望の方は、次頁の申し込み用紙にご記入のうえ、代金とともに下記の宛先まで現金書留にてお送りください(領収書ご希望の方は、その旨を申し込み用紙に明記してください)。

●申し込み締め切り：1996年9月30日(当日消印有効)

●宛先：〒170 東京都豊島区巣鴨 1-14-2

CQ 出版(株) デザイン ウェーブ

OpenDesign No.1 SU 係

*発送までに2週間ほどかかることがあります。


```

int scsi_messgin(int dn, uchar *buf, int siz)
{
    cpu.h.ah = 0x1f;
    cpu.h.al = 0xc0 + (dn & 0xf);
    cpu.x.dx = 0x0080;
    cpu.x.cx = siz;
    cpu.x.bx = FP_OFF(buf);
    seg.es = FP_SEG(buf);
    seg.ds = data_seg;
    int86x(SCSI BIOS, &cpu, &rsi, &seg);
    return(rsi.x.cflag);
}

int scsi_datain(int dn, uchar *buf, int siz)
{
    cpu.h.ah = 0x19;
    cpu.h.al = 0xc0 + (dn & 0xf);
    cpu.x.dx = 0x0080;
    cpu.x.cx = siz;
    cpu.x.bx = FP_OFF(buf);
    seg.es = FP_SEG(buf);
    seg.ds = data_seg;
    int86x(SCSI BIOS, &cpu, &rsi, &seg);
    return(rsi.x.cflag);
}

int scsi_dataout(int dn, uchar *buf, int siz)
{
    cpu.h.ah = 0x18;
    cpu.h.al = 0xc0 + (dn & 0xf);
    cpu.x.dx = 0x0080;
    cpu.x.cx = siz;
    cpu.x.bx = FP_OFF(buf);
    seg.es = FP_SEG(buf);
    seg.ds = data_seg;
    int86x(SCSI BIOS, &cpu, &rsi, &seg);
    return(rsi.x.cflag);
}

int scsi_negate_ack(int dn)
{
    cpu.h.ah = 0x03;
    cpu.h.al = 0xc0 + (dn & 0xf);
    seg.es = seg.ds = data_seg;
    int86x(SCSI BIOS, &cpu, &rsi, &seg);
    return(rsi.x.cflag);
}

/*
 * ASPI SCSI BIOS
 */
int aspi_initialize(void)
{
    int i;
    if(aspi_init() == FALSE)
        return(FALSE);
    memset(&ha_inquiry, 0, sizeof(struct Host_Adapter_Inquiry));
    for(i = 0; i < 2; i++) {
        ha_inquiry.rblock.Host_Adapter_Number = (uchar)i;
        aspi_exec((void *)&ha_inquiry);
        if(ha_inquiry.rblock.Status == TRUE)
            break;
    }
    if(i == 2)
        return(FALSE);
    scsi_io.rblock.Command_Code = 2;
    scsi_io.rblock.Host_Adapter_Number =
        ha_inquiry.rblock.Host_Adapter_Number;
    return(TRUE);
}

int aspi_bios(int id, struct SCSI_PACKET *p)
{
    scsi_io.rblock.SCSI_Request_Flags = p->req_flags;
    scsi_io.Target_ID = (uchar)id;
    scsi_io.LUN = 0;
    scsi_io.Data_Allocation_Length = (long)p->data_length;
    scsi_io.Sense_Allocation_Length = 22;
    scsi_io.Data_Buffer_Pointer = p->buffer;
    scsi_io.SCSI_CDB_Length = p->cdb_length;
    memmove(scsi_io.CDB, p->cdb, p->cdb_length);
    aspi_exec(&scsi_io);

    /* コマンド終了待ち */
    while(scsi_io.rblock.Status == FALSE);

    p->status = scsi_io.Target_Status;
    if(scsi_io.Target_Status == STS_CHK_CONDITION)
        memmove(p->sense, scsi_io.CDB+scsi_io.SCSI_CDB_Length, 22);
    if(scsi_io.Host_Adapter_Status == TIMEOUT) {
        printf("継続処理不能なエラーが発生しました\n");
        return(FAIL);
    }
    return(NORMAL);
}

```

■オープンデザイン No.1 SUディスク・サービス申し込み用紙

OD9401

送り先ご住所：〒

▶ 頒布希望メディア

(どちらかに印をしてください)

☐ 5.25" 2HD ☐ 5.25" 2DD☐ 3.5" 2HD ☐ 3.5" 2DDふりがな
お名前：

TEL ()

会員番号	CQ								
------	----	--	--	--	--	--	--	--	--

- 資料請求会員番号でもお申し込みいただけます。会員番号のみを記入された方には、登録住所へ送付いたします。会員番号と住所を両方記入された方には、記入された住所に送付いたします。
- 希望メディアに印のない場合は、5.25"2HD をお送りいたします。

ソース・リスト2 ASPIASM.ASM

```

MODULE NAME: ASPIASM (For MS-C v5.0 & 6.0)

機能: Adaptec, Inc. 製 Advanced SCSI Programming Interface(ASPI)用

    aspi_init()    :ASPIの初期処理
    aspi_exec()    :ASPIの実行

    CREATION DATE:1/21/93
    Version:1.1(1/21/93)
    AUTHOR:Y. MAXI

TRUE    = 1
FALSE   = 0

argv     struc
    bpsave     dw    ?    :bp save
    ret_ip      dw    ?    :return address ip
    ret_cs      dw    ?    :return address cs
    srb_off     dw    ?    :SRB Pointer(offset)
    srb_seg     dw    ?    :SRB Pointer(segment)
argv     ends

;----- DATA SEGMENT -----
DGROUP   group mdata
mdata    segment word public 'DATA'
    assume ds:DGROUP
    public _aspi_entry

SCSIMgrString db    "SCSIMGRS"
    dw    0
_aspi_entry db    4 dup(?)

mdata    ends
;----- CODE SEGMENT -----
_TEXT    segment byte public 'CODE'
    assume cs:_TEXT, ds:DGROUP
    public _aspi_init
    public _aspi_exec

_aspi_init proc far
    push bp
    mov bp, sp
    push ds
    push bx
    push cx
    push dx

```

```

    lea dx, SCSIMgrString
    mov ax, 3d00h
    int 21h          :Open ASPI Manager
    jc  NoASPIManager :Branch if none found
    push ax          :Save ASPI File Handle

    mov bx, ax       :BX = File Handle
    mov ax, 4402h
    lea dx, _aspi_entry :Store entry point here
    mov cx, 4        :Four bytes to transfer
    int 21h
    jc  NoASPIManager :Branch if ioctl done
    mov ah, 3eh
    pop bx           :BX = File Handle
    int 21h          :Close ASPI Manager

    mov ax, TRUE
    jmp aspi_init_end
NoASPIManager: mov ax, FALSE
aspi_init_end:

    pop dx
    pop cx
    pop bx
    pop ds
    pop bp
    ret

_aspi_init endp

_aspi_exec proc far
    push bp
    mov bp, sp
    push ds
    push es
    push ax
    push bx
    les ax, dword ptr[bp].srb_off
    push es
    push ax
    lea bx, _aspi_entry
    call dword ptr[bx]
    add sp, 4
    pop bx
    pop ax
    pop es
    pop ds
    pop bp
    ret

_aspi_exec endp

_TEXT    ends
end

```

SU.EXE は MS-C ver 5.10, ver 6.0 のラージ・モデルでコンパイルし, Microsoft Assembler ver 5.10 でアセンブルし, Microsoft Linker ver 5.10 でリンクしました。コンパイル手順を右図に示します。

図 SCSI ユーティリティ SU のコンパイル手順

```

A>masm aspiasm...
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988. All rights reserved.

46828 + 244177 Bytes symbol space free

0 Warning Errors
0 Severe Errors

A>
A>cl /AL /c /J /Zp /Gs /BICIL su.c >su.err
Microsoft (R) C Optimizing Compiler Version 6.00A
Copyright (C) Microsoft Corp 1984-1990. All right reserved.
A>

A>link
Microsoft (R) Segmented-Executable Linker Version 5.10
Copyright (C) Microsoft Corp 1984-1990. All rights reserved.

Object Modules [OBJ]: su+aspiasm
Run File [su.exe]: su
List File [NUL.MAP]: su
Libraries [LIB]: libbce.lib
Definitions File [NUL.DEF]:

```


汎用計測器と併用するSCSI簡易ツールの製作

有吉 和久

汎用の周波数カウンタやロジック・アナライザを利用して、特定の情報転送フェーズにおける転送データ数やそのデータ内容を測定するための簡易ツールを紹介する(図1)。

■ SCSI のデータ転送タイミング

SCSI のデータ転送タイミングを図2に示します。非同期と同期の両方の転送方式で、SCSI バス上のデータが確定しているタイミングは、イニシエータ(I)からターゲット(T)へのデータ転送時(以下、I→T時)はACK 信号="L"の期間であり、ターゲットからイニシエータへのデータ転送時(以下、T→I時)はREQ 信号="L"の期間であることがわかります。また、表1のように、情報転送フェーズの種類は、RST 信号、SEL 信号、BSY 信号、I/O 信号、C/D 信号、MSG 信

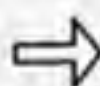
号の論理状態で区別できます。

■ 回路図の説明

図3に、簡易ツールの回路図を示します。製作が簡単のように、LS-TTL を3個で実現しています。CN1とCN2の先には、SCSI 信号の取り出しや計測器へ接続が容易なようにIC クリップを使用します。情報転送フェーズの種類を選択するために8ビット・マグニチュード・コンパレータの74HC688を使用して、そのP端子に入力されたRST、SEL、BSY、I/O、C/D、MSGの各信号論理とQ端子に入力された設定論理を比較します。回路図では、I/O、C/D、MSGの3信号が、それぞれP3、P4、P5の端子に入力され、4連DIPスイッチ S_i のSW1、SW2、SW3の各設定状態と比較されています(表2)。

図1
SCSI 簡易ツールの概要

シングルエンド
SCSI バス



簡易ツール
情報転送フェーズの設定 SW



周波数カウンタ
ロジック・アナライザ

図2 SCSI のデータ転送タイミング

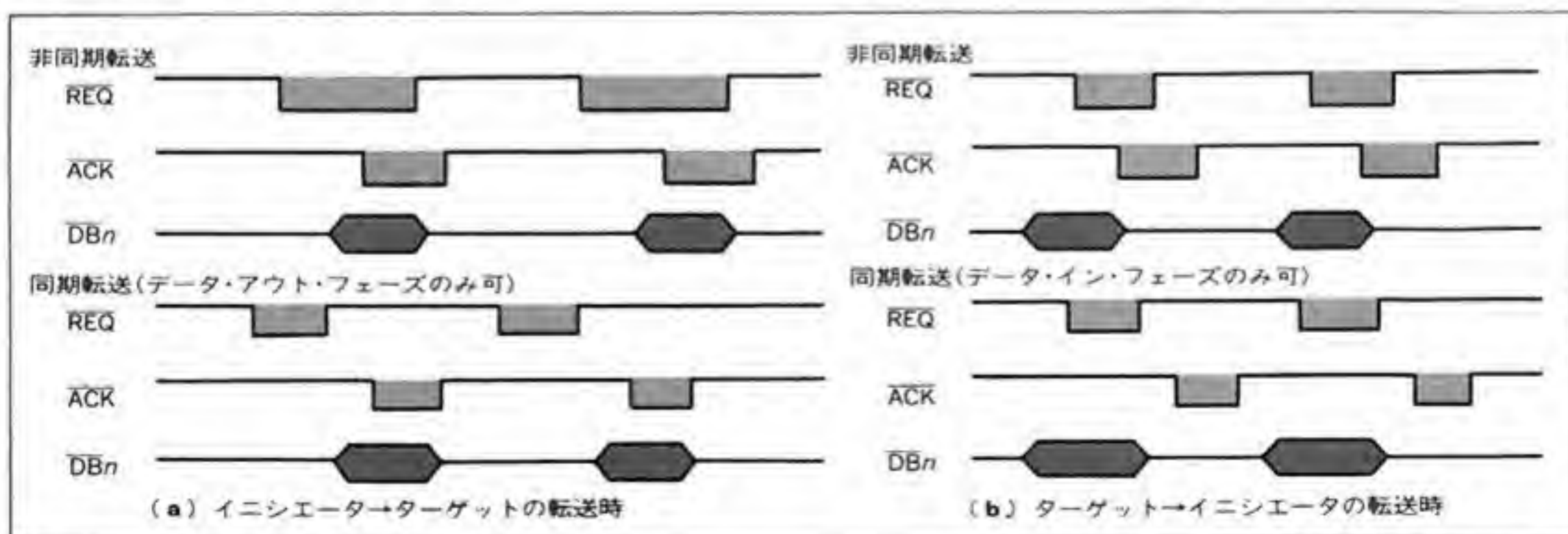
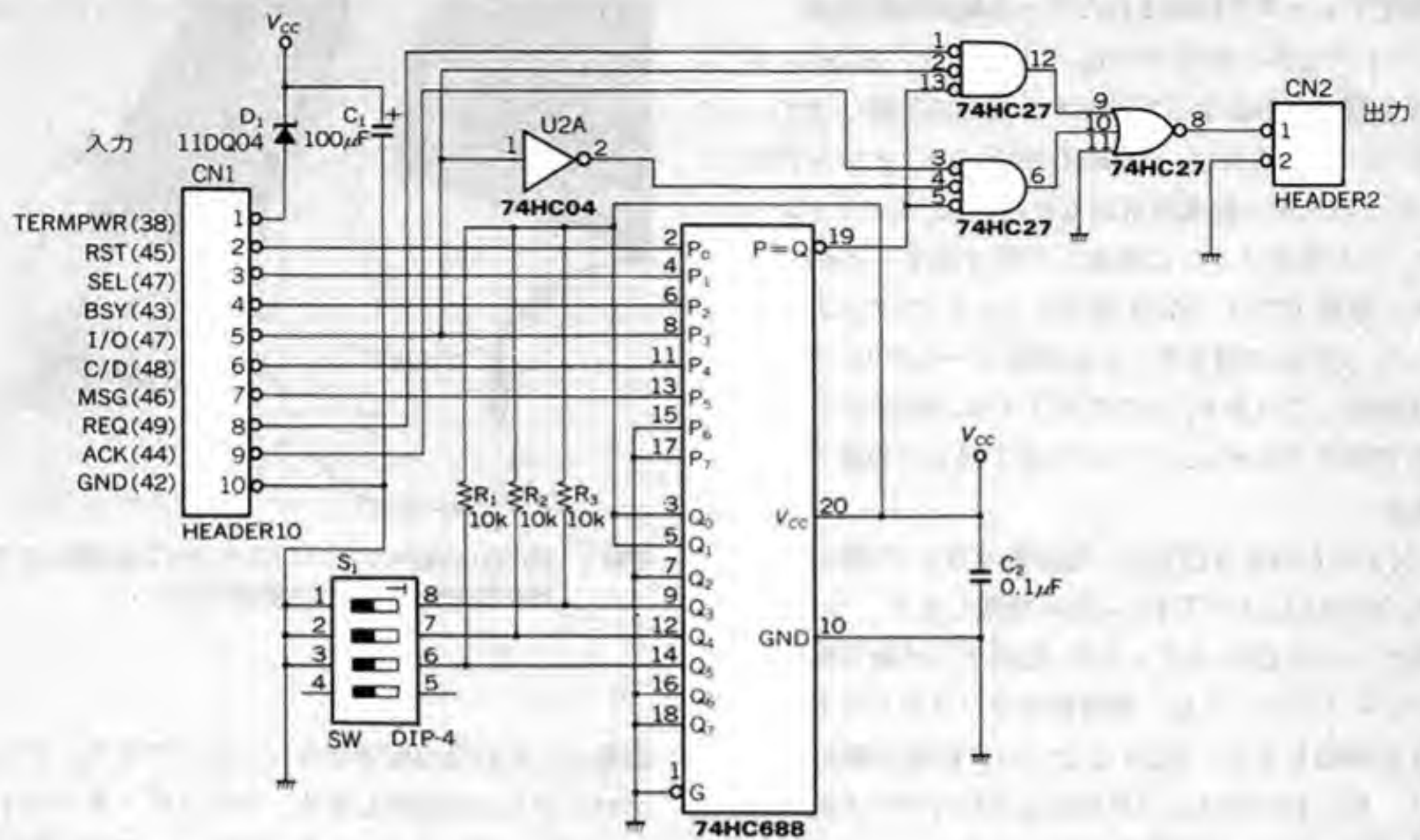


図3 簡易ツールの回路図



このDIPスイッチ設定で指定されるフェーズ状態と、この3信号の論理が一致したとき、P=Q端子に“L”レベルが出力されます。指定フェーズの間中は、P=Q端子が“L”レベルとなるので、その期間に発生するREQ、ACKの信号を取り出せばよいことになります。転送データ方向(I→T、T→I)は、I/O信号で区別できるので、I/O=“L”のときはREQのパルスが、I/O=“H”のときはACKのパルスが有効となるように、3入力NORの74HC27とインバータの74HC04を使用してP=Q端子の出力をゲートしていま

す。

これで、CN2からは、指定のフェーズで、かつSCSIバス上に有効データがあるときに“L”レベルのパルスが出力されることになります。また、パルス幅は、REQまたはACKのパルス幅とほぼ等しくなります。なお、CN1の1ピンからVccに接続されたダイオードは、この簡易ツールの電源をSCSIコネクタのターミネーション・パワーの供給ライン(38ピン)から取り出すときは、省略してもかまいません。

表1 情報転送フェーズ時のSCSI信号論理

フェーズ名称	RST	SEL	BSY	I/O	C/D	MSG
ステータス	H	H	L	L	L	H
データ・イン	H	H	L	L	H	H
メッセージ・イン	H	H	L	L	L	L
未定義イン	H	H	L	L	H	L
コマンド・アウト	H	H	L	H	L	H
データ・アウト	H	H	L	H	H	H
メッセージ・アウト	H	H	L	H	L	L
未定義アウト	H	H	L	H	H	L

表2 選択フェーズとDIPスイッチS_iの設定

フェーズ名称	DIPスイッチS _i		
	SW 1	SW 2	SW 3
ステータス	ON	ON	OFF
データ・イン	ON	OFF	OFF
メッセージ・イン	ON	ON	ON
未定義イン	ON	OFF	ON
コマンド・アウト	OFF	ON	OFF
データ・アウト	OFF	OFF	OFF
メッセージ・アウト	OFF	ON	ON
未定義アウト	OFF	OFF	ON

■ 使い方

▶ 特定フェーズで転送されたデータ数の測定方法

CN1 から出た信号用の IC クリップを、SCSI コネクタ裏側の導体部などで対応する信号に接続します。簡易ツールの電源は、TERMPWR の IC クリップを SCSI コネクタの終端抵抗用電源(ターミネーション・パワー)の供給ラインに接続して取り出すことができます。写真1では、SCSI 信号チェック・アダプタを利用して、SCSI の信号ラインに簡易ツールの IC クリップを接続しています。このアダプタは、SCSI の各信号とアダプタ上のチェック・ピンが1対1で接続されています。

CN2 からの出力信号は、周波数カウンタ(測定周波数 10 MHz 以上)の TTL 入力に接続します。つぎに、簡易ツールの DIP スイッチを、転送データ数を測定するフェーズにセットし、周波数カウンタをクリアして測定を開始します。SCSI 上でデータ転送が開始されれば、セットしたフェーズで転送されたデータ数のみが周波数カウンタに積算表示されます。

▶ 特定フェーズで転送されたデータの解析方法

10 MHz 以上の外部クロック入力でステート解析できるロジック・アナライザがあれば、簡易ツールの DIP スイッチで指定したフェーズ中に転送されたデータの内容を解析することができます。今回製作した簡易ツールの出力信号は、各フェーズでの転送データが SCSI バス上に確実にのっているタイミングを表しています。SCSI データ・バス信号(DB0~DB7)をロジック・アナライザの8本の入力チャンネルに接続し、



写真1 SCSIの信号ラインにICクリップを接続してデータ数を周波数カウンタで積算表示する

簡易ツールの出力信号をロジック・アナライザの外部クロック入力に接続します。ロジック・アナライザのサンプリング・クロックを外部クロックの立ち下がりに選択して測定を開始すれば、データ・バス信号の状態をこの出力信号のタイミングでサンプリングし、ロジック・アナライザのメモリに記録できます。あとは、メモリの内容をロジック・アナライザのステート表示機能で解析します。ただし、SCSIは負論理であるため、ステート表示される16進数をビット反転した値が実際のデータ内容となる点を注意して解析しなければなりません。

ありよし・かずひさ 積水化学工業株式会社応用電子研究所

第6章

SCSIトラブル相談室

失敗例に学ぶSCSIシステム活用/構築ノウハウ

回答者 有吉和久/清水哲夫/真樹美智/吉田浩幸

SCSIの-1と-2の混在使用がうまくいかない原因は?/FAST SCSIの高速でトラブルが/55ボードでHD以外をつなげるか、などユーザ側からのトラブル相談をはじめ、プロトコルやハードウェアにまつわ

る、装置設計の際に出会う問題点まで——代表的なケース・スタディを通して学ぶ、SCSIシステム構築のノウハウと活用のポイント。

(編集部)

Q

SCSI-1とSCSI-2を混在使用する時の注意点は?

A SCSI-1規格の対応機器とSCSI-2規格の対応機器を混在してSCSIバスに接続するときに、今後とくに注意を要することとして、コネクタの信号アサイン中で12, 14, 37, 39ピンの定義が異なる点があげられます。SCSIでは、12, 14, 37, 39ピンがGNDとして規定されていたのに対して、SCSI-2では、Reserved(予約)となっており、これらのピンをオープンにすることを要求しています。

■ SCSI-2 コネクタの信号アサイン(12, 14, 37, 39ピン)に注意!

SCSI-2規格は、SCSI機器との混在利用も可能なように決められているはずですから、これらのピンは、SCSI機器が接続された時点でGNDにショートされることとなります。せっかくこの4本のピンを「リザーブ」に規定して将来使用しようとしても、あまり有効に利用できるとは思えません。

この信号アサインが規定された背景となるものかどうかは別にして、一つの事例を紹介しましょう。

ある外国製ワークステーションに標準装備されているSCSIは、SCSI規格で推奨された形状のコネクタを採用しながら、その12, 37ピンが本来のGNDではなく、そのメーカー独自の仕様で利用されているようでした。おそらく、そのメーカー製のSCSI機器だけが接続さ

れていることを判定するために利用しているのでしょうが、標準インターフェースとしてSCSIを考えたと

図1 ロサブ25ピン・コネクタを採用したアップルのSCSIの例[※] 旧アップル・コンピュータではNC(未接続)

Pin No.	Signal Name	Signal Description
1	REQ/	Request
2	MSG/	Message
3	I/O/	Input/Output
4	RST/	Reset
5	ACK/	Acknowledge
6	BSY/	Busy
7	GND	Signal ground
8	DB0/	Data Bit 0
9	GND	Signal ground
10	DB3/	Data Bit 3
11	DB5/	Data Bit 5
12	DB6/	Data Bit 6
13	DB7/	Data Bit 7
14	GND	Signal ground
15	C/D/	Control/Data
16	GND	Signal ground
17	ATN/	Attention
18	GND	Signal ground
19	SEL/	Select
20	DBP/	Data bit -
21	DB1/	Data Bit 1
22	DB2/	Data Bit 2
23	DB4/	Data Bit 4
24	GND	Signal ground
25	TERMPWR	Terminator power ※1

Connector Type: DB-25 Male

CAUTION: This port uses the same type of connector as a standard RS-232 serial interface, but is electrically very different. DO NOT connect any RS-232 device to this connector. Doing so can result in damage to both the device and the Macintosh Plus.

RS-232-Cと同じ形状のコネクタを使用しているが、けっしてRS-232と接続してはいけなく注意している

アップル・コンピュータ SCSIポート

13.....1
25.....14

DB-25

き、あまり感心できません。

SCSI 用計測器(モデル名: SC/1, SC/2)は、アンフェノール型 50 ピン・コネクタを採用している SCSI 機器の接続判定用に、コネクタの 12, 37, 39 ピンが GND であるかどうかで判定していますが、このメーカーの製品をモニタ観測するために、計測器の専用接続ケーブルの配線を一部変更して、12 ピンと 37 ピンに相当する計測器入力信号を GND に接続する改造が必要になってしまいました。

■ 50 ピン以外の特殊コネクタにも注意!

このことがわかるまで、その対応に苦労したことはいうまでもありません。50 ピン以外の特殊なコネクタを採用した SCSI 製品では、その機器のメーカーが独自の信号アサインを決めている場合もありますが(図 1)。

SCSI 規格の推奨コネクタと同じアンフェノール型 50 ピン・コネクタでも、この事例のように独自の信号アサインの製品が市場に混じっている可能性があることにも注意しなければなりません。

また、SCSI-2 規格では、この信号アサインの変更以外にも、ハーフ・ピッチのピン型コネクタが推奨コネクタとして追加規定されました。しかし、現在日本で開発されるほとんどの SCSI 機器が、大手パソコン・メーカーが採用したハーフ・ピッチのベローズ型コネクタを採用する傾向にあり、事実上の日本標準となっています。これらのことを考えていくと、SCSI は、規格化された標準インターフェースでありながら、いちばん基本的なコネクタ形状や信号アサインがきっちりと規格化されず「推奨」扱いであった点に問題を強く感じています。

(有吉和久)

Q 従来から使用していた SCSI ハード・ディスクの容量が不足したため、新型の SCSI-2 対応ディスクを追加したところ、動作が不安定になるのだが?

A この例も SCSI-1 の旧型と SCSI-2 の新型のケーブルの混在使用によるトラブルだと思われます。

SCSI 機器の接続に利用されている SCSI ケーブルには、非常に多くの種類があります。コネクタ形状やケーブルの長さなどの物理的な違いだけでなく、ケーブルそのものの電気的な特性にもかなり大きな差があります。

初期の SCSI 機器は、伝送速度が 1 M バイト/秒程度までのものが多く、また、デジィ・チェーンで接続される台数もイニシエータとなる SCSI ボードとハード・ディスクの 2 台だけの場合がほとんどだったため、極端な言い方をすれば、機器のコネクタ形状に合

ったケーブルであれば、どんなものでも問題を起こすことなく使用できました。

しかし、今日のように転送速度の高速化した SCSI 機器が普及し、SCSI バス上に複数台の SCSI 機器が接続されるようになると、SCSI ケーブルの電気的な特性による問題が発生する可能性が大きくなりました。

このトラブルの場合、2 台のハード・ディスクは同じメーカー製であり、パソコン側にセットした SCSI ボードも、追加した新型ハード・ディスクに付属していたものを使用していました(図 2)。

■ 接続位置を変えたら正常になった

また、新型のハード・ディスクの取扱説明書には、

図 2 問題が起こった接続

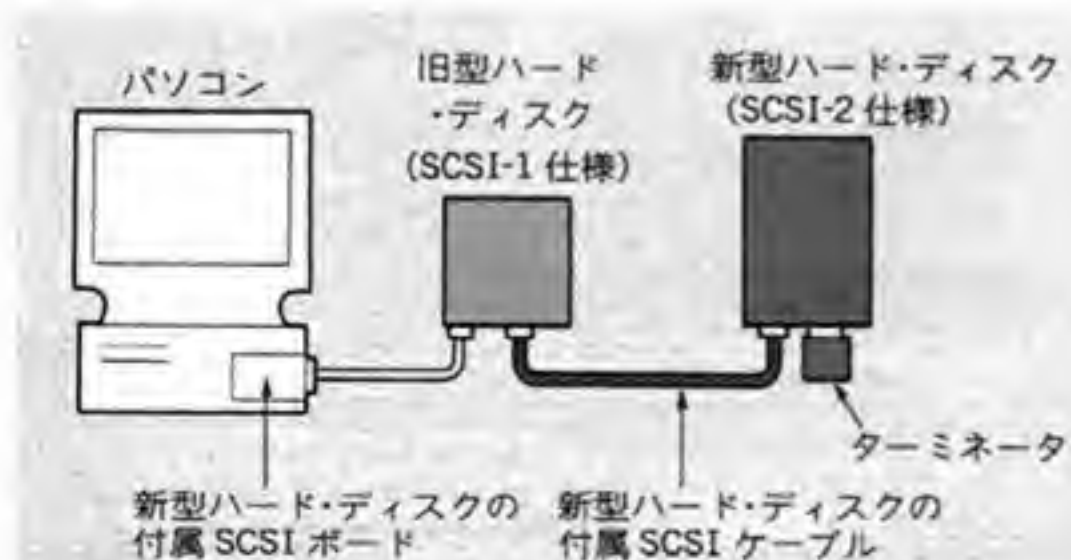
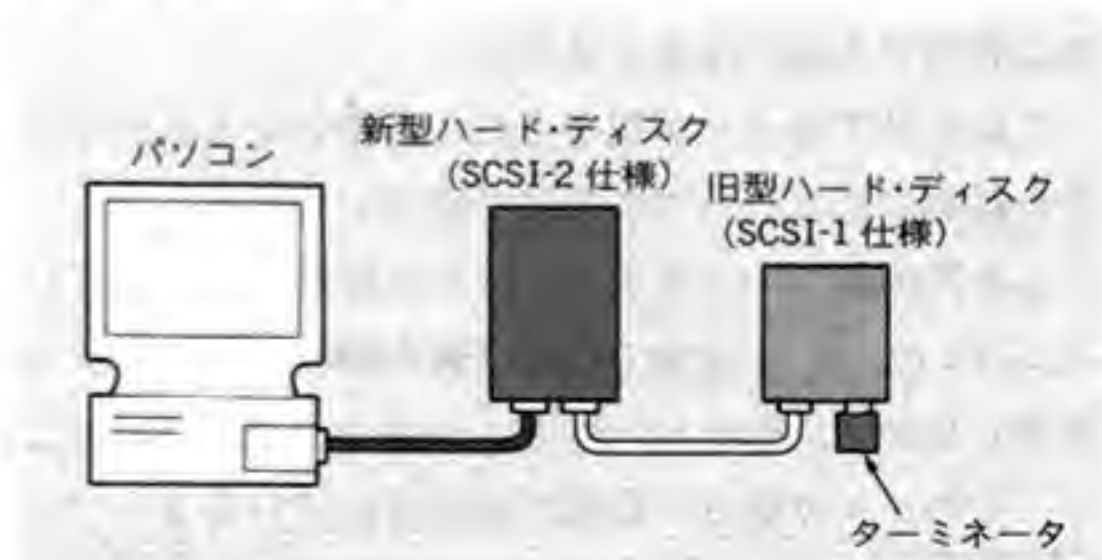


図 3 改善された接続



「旧モデルとの同時使用が可能」と書いてあり、ID の設定やターミネータの接続も間違いなく行われていました。あれこれ悩んだ末に、たまたま新型のハード・ディスクと旧型ハード・ディスクの SCSI バス上での接続位置を変更してみると、今までの不安定な動作が嘘のように解消されてしまいました(図 3)。

本来、SCSI バス上にデジィ・チェーンで接続する機器の位置関係は任意であるはずなのですが、実際はこの例のようにおかしな現象が起こってしまいます。原因は、デジィ・チェーン接続に複数利用される SCSI ケーブルのうちの何本かの電気的特性、とくにケーブルのインピーダンス(交流的な抵抗値)が、SCSI 規格の規定を満足していないためと考えられます。

■ ケーブルの低インピーダンスに注意

規格では、ケーブルの電気的特性についてつぎのように規定しています。

- ① 特性インピーダンス：90～132 Ω
- ② 信号減衰率：0.095 dB/m(at 5 MHz) 最大
- ③ 伝搬遅延：0.20 ns/m 最大
- ④ DC 抵抗：0.23 Ω /m(at 20°C) 最大

このうち一般的なキャブタイヤのツイストペア・シ

ールド・ケーブルを SCSI ケーブルとして利用したときに問題になるのは①の条件であり、この場合、たいてい 50～80 Ω 程度のケーブルができあがってしまいます。初期型の SCSI 機器に付属する SCSI ケーブルには、このようにインピーダンスの低いものがけっこうあります。

今回のトラブル例も、低インピーダンスの旧ケーブルと新型ハード・ディスクに付属していた高インピーダンスの新ケーブルの境目で信号の反射が発生したためです。最初の設置状態で動作が不安定となっていたものが、新型ハード・ディスクをパソコンに近い側に設置したことで、FAST SCSI 仕様で高速転送される距離が短くなったために、信号の反射が発生しながらも、見かけ上は問題なく動作したものと考えられます。

SCSI-2 規格で規定された高速同期転送時の信号タイミング(最高速度 10 M バイト/秒相当)が、ひと昔前の 16 ビット CPU バスのタイミングよりクリティカルであり、しかも信号線が最大 6 m(シングルエンド型インターフェース時)もケーブルで引き回される可能性があることに注意して、慎重にケーブルやターミネータなどの伝送系の特性を考慮する必要があります。

(有吉和久)

Q

FAST SCSI-2 の 10 M バイト/秒でデータ転送すると、とくに複数台のターゲット・デバイスを接続したときにエラーが起きやすいが?

A SCSI-2 の FAST オプションは、原則としてディファレンシャル・バスを採用したシステムのためのものです。SCSI-2 の規格書では、シングルエンド・バスでの FAST オプションは推奨しないと記述されています(Use of single-ended drivers and receivers with the fast synchronous data transfer option is not recommended.)。SCSI-3 のワーキング・ドラフト(X3T9.2/855D Revision 12b 1993/6/11)では、5 M バイト/秒以上のデータ転送を行う場合には、シングルエンド・バスの最大ケーブル長を現行の 6 m から 3 m に半減させているほどです(ただし、それ以下のスピードの場合には 6 m まで許される)。

■ 10 M バイト/秒同期転送に成功する三つのポイント

要するに、シングルエンド・バスで 10 M バイト/秒の同期転送を行いたい人は、自分で責任をもちなさい

ということなのですが、「よし、責任をもとう」という方のためにアドバイスをいくつか挙げておきましょう。

(1) ターミネータはバスの両端ともアクティブ・ターミネータを使用する。

(2) よいケーブルをできるだけ短く使う。よいケーブルと悪いケーブルの差は経験からいっても意外と大きく、決して侮れません。シールドなし、ツイストなしのフラット・ケーブルなどは論外です。最近では、この手の問題が増えてきたのか、SCSI-2 用の「ハイ・インピーダンス・ケーブル」なるものが出まわっています。これらの実態がどういうものか、詳しく調べていないので迂闊なことはいえませんが、特性インピーダンスを高めにし、REQ/ACK 線を中心部にもってくるなど工夫を凝らしてあるようで、試す価値は十分にあります。

(3) デジィ・チェーンする場合には、違う種類のケーブルを混在させない。

(清水哲夫)

Q

98 の 55 ボードにハード・ディスクや MO 以外の SCSI 機器を接続したいが、問題なく立ち上がるか？

A 立ち上がるかどうかを説明するための予備知識として、一般的な SCSI 機器とイニシエータ(ホスト)の立ち上がりシーケンス(通信手順)から説明しましょう。大きく分けて、SCSI 機器はハード・ディスクのように媒体の取り外しができないものと、媒体の取り外しができるもの(MO、CD-ROM、テープ・ドライブなど)とに分かれます。

■ 媒体の取り外しが可能かどうか

その違いによりターゲットがイニシエータに報告する Unit Attention が若干変わってきます。媒体の取り外しが可能(リムーバブル)であった場合、ターゲットの状態は Ready(装着中)と、Not Ready(未装着)とに分かれます。また、Ready であっても媒体を交換直後の場合、ホストのファイル・システムがその媒体の容量なりディレクトリなりを知る必要が発生します。これらの状態をターゲットがホスト(イニシエータ)に知らせるため、SCSI 規格では Unit Attention 条件が定義されています。

媒体の取り外しが可能かどうかの判断や、機器の種別は、Inquiry データで知ることができるので、普通、ホストがセレクションに応じたターゲットに対して初めに発行するコマンドは Inquiry です。

■ 媒体の取り外せないデバイスの立ち上がりシーケンス

まず、媒体が取り外せないターゲットの場合の立ち上がりシーケンスを示しましょう(以下、I: イニシエータ、T: ターゲットとする)。

(1) I: ターゲットに対して Inquiry コマンドを発行する。

なお Inquiry、Request Sense コマンドは、媒体を装着しているか、いないかに関わらず、発行できる(CDB 中の予約ビット・エラーやパリティ・エラーでない限りチェック・コンディション・ステータスを返さない)コマンドです。

(2) T: イニシエータに対して Inquiry データを返す(Good ステータスを返す)。

(3) I: Inquiry データの周辺機器種別(バイト・ポイ

ンタ=0、ビット・ポインタ=4、ビット・レングス=5、以降このポインタを 0-4-5 と表記する)か、RMB: Removable(1-7-1)がオンで取り外し可能媒体であることを記憶しておく。

ただし、PC-9801 の場合 HD、MO 以外の管理の記憶域はありません。イニシエータ(ホスト)によっては、サポートしていない機器に対してこれ以上のコマンドを発行しないものもあります。

(4) I: ターゲットに対して Test Unit Ready コマンドを発行する。

(5) T: イニシエータに対してチェック・コンディション・ステータス(02h)で応答する。

(6) I: ターゲットに対して Request Sense コマンドを発行し、チェック・コンディション・ステータス(02h)の原因をターゲットに問い合わせる。

(7) T: イニシエータに対して Request Sense データを送る(Good ステータスを返す)。このときセンス・データは以下の内容となる。

Sense Key(2-3-4): 6 Unit Attention

ASC(12-7-8): 29h

ASCQ(13-7-8): 00h POWER ON RESET or BUS DEVICE RESET OCCURRED

<注意>システムの立ち上がりは、普通、電源のオンから始まります。このためターゲットは、自身がパワー・オン・リセットから始まったことをイニシエータに知らせるために、Unit Attention を報告しなければなりません。また、SCSI バス・ラインの RST(リセット信号)によりリセットされた場合でも同様です。

(8) I: ターゲットに対して Test Unit Ready コマンドを発行する。

(9) T: イニシエータに対して Good ステータス(00h)で応答する。

■ 媒体取り外し可能ターゲットの立ち上がりシーケンス

以上が、媒体が取り外し不可のターゲットの立ち上がりシーケンスです。取り外し可能ターゲットの場合は、(9)の応答が変わり、さらに以下のシーケンスが加わります。

(9) T: イニシエータに対してチェック・コンディション・ステータス(02h)で応答する。

(10) I: ターゲットに対して Request Sense コマンドを発行し、チェック・コンディション・ステータス(02h)の原因をターゲットに問い合わせる。

— もし、この時点で媒体が装着されていた場合 —

(11)-1 T: イニシエータに対して Request Sense データを送る(Good ステータスを返す)。このときセンス・データは以下の内容となる。

Sense Key(2-3-4): 6 Unit Attention

ASC(12-7-8): 28h

ASCQ(13-7-8): 00h NOT READY TO READY TRANSITION, MEDIUM MAY HAVE CHANGED

<注意>媒体が取り外されている状態から、装着された状態になったとき、ターゲットは、この Unit Attention を報告しなければなりません。これは、電源オンのときに装着されていても同様です。イニシエータはこのセンスが返されると、MO であればローダを読むか(立ち上がりディスクであれば)、ディレクトリを読む(読み直す)かします。

— もし、この時点で媒体が装着されていない場合 —

(11)-2 T: イニシエータに対して Request Sense データを送る(Good ステータスを返す)。このときセンス・データは以下の内容となる。

Sense Key(2-3-4): 2 Not Ready

ASC(12-7-8): 3Ah

ASCQ(13-7-8): 00h MEDIUM NOT PRESENT

<注意>この Not Ready の状態から、装着された状態になったとき DOS の DIR などのトリガでイニシエータは、(8)からの通信を繰り返します。

以上が、一般的な SCSI 機器の立ち上がりシーケンスです。以降のシーケンスは書き込み/読み込み可能媒体の場合、Macintosh, FMR などのように容量やパーティション情報がインストール時にメディアに書き込まれているホストは、その情報を Read コマンドで読

みます。PC-9801, AT などシリンダ数、セクタ/トラックなど諸元情報の必要なホストは、パーティション情報を読む前に Read Capacity コマンドや Mode Sense コマンドを発行します。

■ 55 ボードと HDD, MO 以外の SCSI 機器の接続

PC-9801-55U に関して、Inquiry データの 'NEC' をプロテクトしているのは、ハード・ディスクのみです。MO では、Read Capacity で得られた容量がハード・ディスクと同じように加工されて、諸元エリア(0:460h)に書き込まれます。他の機器では DOS の管理領域のどこにも記録されません。したがってハード・ディスクや MO でなければ、システムの立ち上がりだけでいえば、Inquiry コマンドでの応答(周辺機器種別)がハード・ディスクでなければ接続できます。

さて、立ち上がった後のことですが、DOS がサポートしていない SCSI 機器の場合、デバイス・ドライバ経由か、アプリケーション経由で直接 SCSI BIOS を呼び出さなければなりません。

■ ターゲット・システム開発時の補足

PC-9801-55U は同期転送をサポートしていないので問題はないのですが、サポートしているボード(たとえば ASPI 仕様)に接続するときは注意してください。ターゲット・システムが同期転送を行わないのであれば、Inquiry データの Sync ビット(7-4-1)をオフにしておくか、ボードの設定(ハードまたはソフトで設定する)を非同期モードにするか、またはメッセージ・アウト/インでの同期のネゴシエーションでイジェクトする必要があります。

また、同期禁止モードにしても ASPI はディスコネクトをサポートしているので、セレクション時に ATN を立ててきます。ターゲットはこれに対しメッセージ・アウト・フェーズとし、Identify メッセージ(イニシエータからの C0h)を受け取る必要があります。

(真樹美智)

マイコン技術者スキルアップ事典

エンジニアの仕事地図から理想像までをハード中心に整理した事典。新人教育に最適です。

ハードに強いエンジニアになるためのデータバンク 長嶋洋一著 1,650円(税込) CQ出版社

Q

PC-9801 用のサード・パーティ製 SCSI ハード・ディスク(インストール済み)を PC-9801-92 ボードに接続しようとしたがハングアップしてしまった。インストールしたデータは壊れてもしかたないが、なんとか接続できないか?

A

「4.1 98 の SCSI と 55&92 ボード」(p. 80~)で述べましたが、ボードによって諸元情報エリア(0:460h)に書かれるシリンダ、ヘッド、セクタ/トラックの値が異なるのが、ハングアップの原因です。ハード・ディスクの論理ブロック・アドレス 0 には IPL が書かれています。これは FORMAT.EXE の初期化処理で Format Unit コマンドの実行後に自動的に書かれます。この後、領域確保をすると論理ブロック・アドレス 1 のところにパーティション情報が書かれ、論理ドライブの開始シリンダが特定されます。この IPL、パーティション情報の書かれているセクタは、諸元情報に関わりのない位置なので、92 ボード経由でも、正常にインストールされていると判断され、ブート処理に移ります。ところが、サード・パーティ製ボードでインストールしたときの論理ドライブの開始シリンダが 92 ボードで算出したシリンダ位置と違うため、ブート処理ができずにハングアップしてしまうのです。

■ 対策法

ブート処理を禁止するための方法はつぎに示すようにいくつかあります。これらのいずれかを実行したあ

と 92 ボードに接続して再インストールします。それでも起動しないときは、別の原因です。

- (1) 最初にインストールしたボード(サード・パーティ製)に接続しなおし、FORMAT.EXE で論理ドライブをスリープにする。
- (2) Macintosh などの SCSI 汎用ユーティリティで Format Unit をかける。
- (3) SU.EXE(5 章)のセクタ編集機能を使って IPL のセクタの内容を書き換える(0 データなどに)か、パーティション情報をアクティブ(91h)からスリープ(11h)に書き換える。

ただし、92 ボードで行うときはハード・ディスクの電源を落としてシステムを立ち上げ、その後ハード・ディスクの電源を入れ、ユーティリティを起動する(p. 85 の図 6)。

- (4) あまり勧められない方法ですが、最初にインストールしたボード(サード・パーティ製)に接続しなおし、FORMAT.EXE の初期化を起動し、ハード・ディスクのアクセス・ランプが点灯して 2.3 秒経過したら電源を落とす。初期化を最後まで行くと IPL が書かれる。これは最後の手段。

(真樹美智)

Q

メッセージ・アウト・フェーズでイニシエータから受け取ったメッセージ・バイトがパリティ・エラーだった。ターゲットはどう応答したらよいか?

A

ワークステーション・レベル以上の環境の SCSI バスでは、ATN 信号付きのセレクションの後、ターゲットは通常メッセージ・アウト・フェーズに移行します。そこで最初のメッセージである Identify をイニシエータから受け取るのですが、このときに、バスのノイズ環境に問題があったり、ターミネータの不良や TERMPWR 線の電圧低下、ケーブルの導通不良、さらにはドライバ回路が壊れかかっているなどといったようなことが原因で、このメッセージがパリティ・エラーで受信されることがあります。もちろん、このように電氣的に不安定な状態のときには、SCSI フェーズの任意の時点でパリティ・エラーが発生する可能性があるわけですが、フェ

ーズ遷移の順番からいっても、セレクション・フェーズのつぎに出現するこのメッセージ・アウト・フェーズにおいて、ターゲット側でパリティ・エラーを検出する率が高くなります。

■ ノイズやケーブル不良、ターミネータ不良などを疑ってみる

パリティ・エラーでセレクションされてもターゲットは応答しないことになっていますから、偶然そこを正常に通過すれば、このようなシチュエーションも大いに考えられるわけです。

このような場合には、

- (1) すべてのメッセージを再送してもらう

- (2) 一度の再送でもだめなら、何回かリトライする
- (3) リトライしてもだめなら、Check Condition のステータスを返して、センス・キーに Aborted Command, 追加センス・コードを Message Error にするか、
- (4) あるいは、即座にバス・フリー・フェーズにして

しまう。
 というようなサジェスションが、SCSI-2 の規格書には書かれています。メッセージを再送してもらうためには、ATN 信号がネゲートされた後もメッセージ・アウト・フェーズのまま再び REQ をアサートして、その意図をイニシエータに伝えます。
 (清水哲夫)

Q

存在しないロジカル・ユニット番号に対してイニシエータから Inquiry コマンドがきた。どうすればいい?

A

Inquiry コマンドは、指定したロジカル・ユニット番号(LUN)のデバイスがどういう素性のものかを知るためのコマンドです。通常のアプリケーションでは十中八九、一つのターゲットは一つの LUN(0 番)しかサポートしていないので、イニシエータ側もそれ以上の追求はしないのが普通ですが、律義なイニシエータの場合には、LUN 0 番

の Inquiry が終わった後、続いて 1 番への Inquiry がきたりします。慣れないことをやられると面食らったりするものですが、ずばんなターゲットとしてもここはあせらず騒がず、Inquiry データの第 0 バイト(デバイス・タイプ)を 7Fh(Logical Unit Not Present)にして返します。
 (清水哲夫)

Q

同期転送によるデータ・フェーズの実行で、データ転送数が少なくなってしまう場合や、多くなってしまう場合があるが?

A

このトラブルのようにソフトウェアの問題が考えられる場合、以下に示す順にチェックすることにより問題の早期発見ができます。

- (1) SCSI 制御 LSI に設定したパラメータ
- (2) ターゲットのセクタ数
- (3) 同期転送メッセージでのパラメータ(転送速度、オフセット)
- (4) 制御プログラムでデータ転送を中断しているか?
- (5) SCSI ケーブルの接触不良

データ・フェーズのデータの内容を調べます。その際、データを規則性のあるもの(たとえば 00h~FFh のインクリメント)にすれば解析しやすいと思います。図 4、5 に SCSI バスの状態を示します。図 4 の場合、転送の途中で 1 バイト・データが抜けています。つまりデータの出力側では出したはずのデータが SCSI バスには現れていません。SCSI バス上での問題(SCSI バス上のノイズ)でないとしたら、データの出力側に問題があると判断できます。

■ 転送長が短くなってしまう場合

この現象ではデータの出力側(データ・イン・フェーズの場合にはターゲット装置、データ・アウト・フェーズの場合にはイニシエータ装置)で必要数のデータの出力が行われていない場合と、データの入力側(データ・イン・フェーズの場合にはイニシエータ装置、データ・アウト・フェーズの場合にはターゲット装置)でデータのとりこぼしが発生している場合があります。上記(1)~(5)まで確認できたら、ハードウェア上の問題と判断できます。

どこがおかしいのかを確認するためにはまずデー

図 4 トラブル例——転送の途中で 1 バイト・データが抜けている

MON:DISPLAY		[Area:IT] [DM:3]	
T00000	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
+00016	0A 00 00 00 04 00 00 01	02 03 04 05 06 07 08 09	0A 0B 0C 0D 0E 0F 10 11	12 13 14 15 16 17 18 19	0A 0B 0C 0D 0E 0F 10 11
+00032	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
+00048	1A 1B 1C 1D 1E 1F 20 21	22 23 24 25 26 27 28 29	2A 2B 2C 2D 2E 2F 30 31	32 33 34 35 36 37 38 39	3A 3B 3C 3D 3E 3F 40 41
+00064	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
+00080	4B 4C 4D 4E 4F 50 51 52	53 54 55 56 57 58 59 5A	5B 5C 5D 5E 5F 60 61 62	63 64 65 66 67 68 69 6A	6B 6C 6D 6E 6F 70 71 72
+00096	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
+00112	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
+00128	7B 7C 7D 7E 7F 80 81 82	83 84 85 86 87 88 89 8A	8B 8C 8D 8E 8F 90 91 92	93 94 95 96 97 98 99 9A	9B 9C 9D 9E 9F A0 A1 A2

図5 トラブル例——データ転送が途中で止まっている

MON:DISPLAY	[Area: (T)]	[DM:3]
T00000	0A 00 00 00 04 00 00 01	02 03 04 05 06 07 08 09
+00016	0A 08 0C 0D 0E 0F 10 11	12 13 14 15 16 17 18 19
+00032	1A 1B 1C 1D 1E 1F 20 21	22 23 24 25 26 27 28 29
+00048	2A 2B 2C 2D 2E 2F 30 31	32 33 34 35 36 37 38 39
+00064	3A 3B 3C 3D 3E 3F 40 41	42 43 44 45 46 47 48 49
+00080	4A 4B 4C 4D 4E 4F 50 51	52 53 54 55 56 57 58 59
+00096	5A 5B 5C 5D 5E 5F 60 61	62 63 64 65 66 67 68 69
+00112	6A 6B 6C 6D 6E 6F 70 71	72 73 74 75 76 77 78 79
+00128	7A 7B 7C 7D 7E 7F 80	81 82 83 84 85 86 87 88

図5では転送が途中で止まっています。同期転送実行中に途中で転送が止まってしまう場合、考えられる原因としてはオフセット関係が有力です。最初に設定したオフセットを超える REQ 信号または ACK 信号が出力された場合、入力側ではエラーとするしかありません。

■ 転送長が多くなってしまう場合

この現象ではデータの出力側(データ・イン・フェーズの場合にはターゲット装置、データ・アウト・フェーズの場合にはイニシエータ装置)で必要数以上のデータの出力が行われる場合と、データの入力側(データ・イン・フェーズの場合にはイニシエータ装置、データ・アウト・フェーズの場合にはターゲット装置)でデータを余計に取り込んでいる場合があります。やはり(1)~(5)までの確認が終了した場合、ハードウェア上

図6 トラブル例——転送の途中で同じデータが2回続いている

MON:DISPLAY	[Area: (T)]	[DM:3]
T00000	0A 00 00 00 04 00 00 01	02 03 04 05 06 07 08 09
+00016	0A 08 0C 0D 0E 0F 10 11	12 13 14 15 16 17 18 19
+00032	1A 1B 1C 1D 1E 1F 20 21	22 23 24 25 26 27 28 29
+00048	2A 2B 2C 2D 2E 2F 30 31	32 33 34 35 36 37 38 39
+00064	3A 3B 3C 3D 3E 3F 40 41	42 43 44 45 46 47 48 49
+00080	4A 4B 4C 4D 4E 4F 50 51	52 53 54 55 56 57 58 59
+00096	5A 5B 5C 5D 5E 5F 60 61	62 63 64 65 66 67 68 69
+00112	6A 6B 6C 6D 6E 6F 70 71	72 73 74 75 76 77 78 79
+00128	7A 7B 7C 7D 7E 7F 80	81 82 83 84 85 86 87 88

の問題と判断できます。

どこがおかしいのかを確認するためには、やはりデータ・フェーズのデータの内容の確認が必要です。その際、データを規則性のあるもの(たとえば00h~FFhのインクリメント)にすれば解析しやすいと思います。図6に障害例を示します。この場合、転送の途中で同じデータが2回続いています。つまりデータの出力側が余計なデータを出したことがわかります。

■ SCSI チップと周辺回路を見直す

なぜこのような問題が発生するかについてはいろいろな要因が考えられますが、やはり SCSI コントロール LSI および周辺回路を見直すことが重要です。

本トラブルの場合は、SCSI コントロール LSI の DMA 部と DMA インターフェース間のタイミングに問題があったことが原因でした。(吉田浩幸)

Q

セレクション・フェーズ後のメッセージ・アウト・フェーズで Identify メッセージに続いて同期転送メッセージを送信しようとする、コマンド・フェーズに遷移してしまうか?

A

最初にメッセージ・アウト・フェーズからコマンド・フェーズの間の SCSI バスの動きを確認します。本トラブルの場合、図7、8に示すように2通りの状態が考えられます。

■ 症状

図7では、イニシエータ装置が Identify メッセージ送出後につぎのメッセージを送出するために ATN 信号をアサートしていますが、ターゲット装置がコマンド・フェーズに遷移しています。

図8では、イニシエータ装置が Identify メッセージ

送出後、ACK 信号をネゲートする前に、ATN 信号をネゲートしてしまっているため、ターゲット装置がフェーズをコマンド・フェーズに移行させています。

■ 原因と対策

トラブルの原因が図7で示すように ATN 信号の有無に関わらず、ターゲット装置がコマンド・フェーズにフェーズを遷移させるような場合、つぎの三つの要因が考えられます。

(1) ターゲット装置の制御プログラムが ATN 信号を確認しなかった。

データ・フェーズの最終データを AAh, ステータス・フェーズのデータを 55h にして同様の処理を行うと、ステータス・フェーズでのデータが 00h になりました。これにより少なくともイニシエータ装置が SCSI バスを駆動したままではないことがわかりました。そのときの SCSI バスの状態を図 9 に示します。この図では以下のことがわかります。

- (1) ステータス・フェーズのデータの確定は REQ 信号アサートの前に行われている。
 - (2) データ・フェーズの最終 ACK のネゲートがステータス・フェーズに遷移してから行われている。
- (1) の現象は情報転送フェーズでの通常の状態です (データを出力した後で、REQ 信号また ACK 信号をアサートする)。つまりステータス・フェーズでの異常データはターゲット装置が出力していることになります。
- (2) の現象には問題があります。ターゲット装置は前のフェーズの ACK 信号がネゲートされるまでフェーズをそのままにしておく必要があります。

Q マルチターゲット・システムで、二つ以上のターゲット装置に、ディスコネクト処理を行わせている場合、イニシエータ装置に対してリコネクト処理がされない場合があるが、どうしてか？

A 本トラブルの状況はつぎのようなものでした。

- ・イニシエータ装置：パソコン
- ・ターゲット装置：ハード・ディスク×2台

ディスコネクト中のターゲット装置からリコネクト処理が行われない要因としては以下のものが考えられます。

- (1) ディスコネクト中の処理で、回復不能のエラーが発生した。
- (2) イニシエータ装置から Bus Device Reset, Abort

図9 トラブル例——データ・アウト・フェーズ修了後のステータス・フェーズでステータス・データがデータ・アウト・フェーズの最終データになったときのバスの状態

```

MON:DISPLAY
[Area:(T) ][DM:T]
C:-00006 Cl:T00000 Sample Time:100(nSec)
CltoC:-00000.6(uSec) TtoC:-00000.6(uSec)
      RA D D D D
      B S M C I A R R A E C B B B B P I S
      S E S / / T S E C Q K 0 1 2 3 E D Q
      Y L G D O N T Q X B B PHILPHILPHILPHIL ASCII 0123 E E PHASE
-00006 1 0 0 1 1 0 0 0 0 - - 100 - - - - , - - - 0 - - 0 0 ST
-00005 1 0 0 1 1 0 0 0 0 - - 100 - - - - , - - - 0 - - 0 0 ST
-00004 1 0 0 1 1 0 0 0 0 - - 100 - - - - , - - - 0 - - 0 0 ST
-00003 1 0 0 1 1 0 0 0 0 - - 100 - - - - , - - - 0 - - 0 0 ST
-00002 1 0 0 1 1 0 0 0 0 - - 1AA - - - - , - - - 0 - - 0 0 ST
-00001 1 0 0 1 1 0 0 0 0 - - 1AA - - - - , - - - 0 - - 0 0 ST
T00000 1 0 0 1 1 0 0 1 0 - - 1AA - - - - , - - - 0 - - 0 0 ST
+00001 1 0 0 1 1 0 0 1 0 - - 1AA - - - - , - - - 0 - - 0 0 ST
+00002 1 0 0 1 1 0 0 1 0 - - 1AA - - - - , - - - 0 - - 0 0 ST
+00003 1 0 0 1 1 0 0 1 0 - - 1AA - - - - , - - - 0 - - 0 0 ST
+00004 1 0 0 1 1 0 0 1 0 - - 1AA - - - - , - - - 0 - - 0 0 ST
+00005 1 0 0 1 1 0 0 1 0 - - 1AA - - - - , - - - 0 - - 0 0 ST

```

本トラブルは、ターゲット装置がフェーズを遷移したときに、ACK 信号がアサートしていたため、そのときのデータを SCSI 制御 LSI が取り込んでしまい、ステータス・フェーズでそのデータを出力することにより発生していました。

そこで以下の2箇所の変更を行うことで対処しまし
た。

- (1) ターゲット装置で、フェーズの遷移をさせる前に ACK 信号のネゲートを確認する。
- (2) イニシエータ装置で、データ・フェーズでの ACK 信号のネゲートを行う(実際には ACK 信号のネゲートを忘れていた)。(吉田浩幸)

- メッセージが入力された。
- (3) Reset コンディションが発生した。
- (4) リコネクト処理を行う必要がなかった。またはリコネクト処理を行わなかった。
- (5) アビトレーション・フェーズが成功しない。
- (6) じつはリコネクト処理を行っているが、リコネクトに失敗した。

当該ターゲット装置に対して、Request Sense コマンドを発行してセンス・データを取得することにより上記(1)～(6)までのいずれかを確認することができます(図10)。ターゲット装置のセンス・データがセンス・キー=0B(Aborted Command)、センス・コード=

ズ
(7) その他

■ 結果と対策

結果から先にいいますと、(1)のケーブルの不良でした。自作のケーブルのため、接触不良が発生していました。本トラブルは毎回発生するというものではな

ったため、最初に疑ったケーブルを触っていたところ、SCSI バスが不安定になり、接触不良であることがわかりました。

SCSI バス上に異常がある場合、その頻度に関わりなく上記(1)~(6)の順に確認すれば、より早く原因が判明するでしょう。
(吉田浩幸)

Q

SCSI の TERMPWR 線の電源電圧が低下して転送動作が不安定になってしまうのだが？

A デイジィ・チェーン接続された SCSI 機器のうち、両端に位置する機器には終端抵抗回路(ターミネータ)が必要になります。一般に、SCSI 機器は、デイジィ・チェーンのどの位置に接続されてもかまわないように終端抵抗回路の接続・非接続を選択できます。

SCSI 規格は、終端抵抗回路の接続・非接続に対応した設計を簡単にするため、終端抵抗用電源(ターミネーション・パワー)の供給ラインを SCSI バスの信号線中に TERMPWR 線として規定しています。この TERMPWR 線への電源供給は、デイジィ・チェーン接続されたいずれかの SCSI 機器が行う必要があります。電源を供給する機器は、その機器の動作電源から逆流防止用のショットキ・バリア・ダイオードおよび過電流保護

手段を経由して電源供給を行う方法をとります(図11)。

また、SCSI 規格では、TERMPWR 線への電源供給条件を表1のように規定しています。

■ CMOS-IC の入力回路が問題

表1からわかるように TERMPWR 線への電源供給には、かなりの電源容量が必要であり、電池駆動で動作するような SCSI 機器にとっては、大きな電源負担となります。

そこで、TERMPWR 線の電源供給状態を調べる監視回路と TERMPWR 線への電源供給制御回路を用意することで、自分以外の SCSI 機器がすでに終端抵抗用電源を供給している場合は、TERMPWR 線への電源供給を行わないようにできる機器を設計したこと

図11 一般的なターミネーション・パワー供給回路

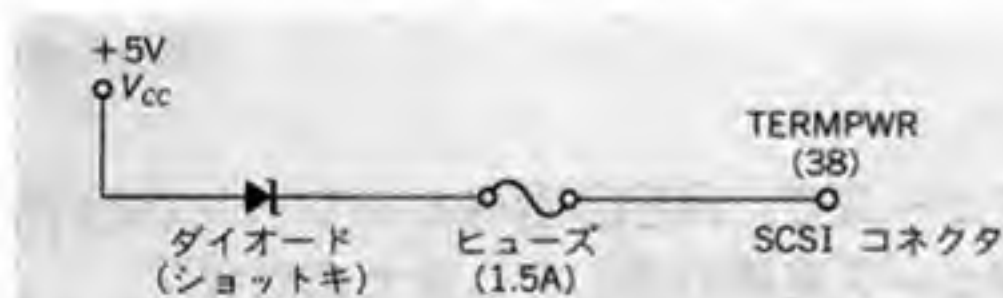


図12 電源 OFF 時に問題が発生した回路

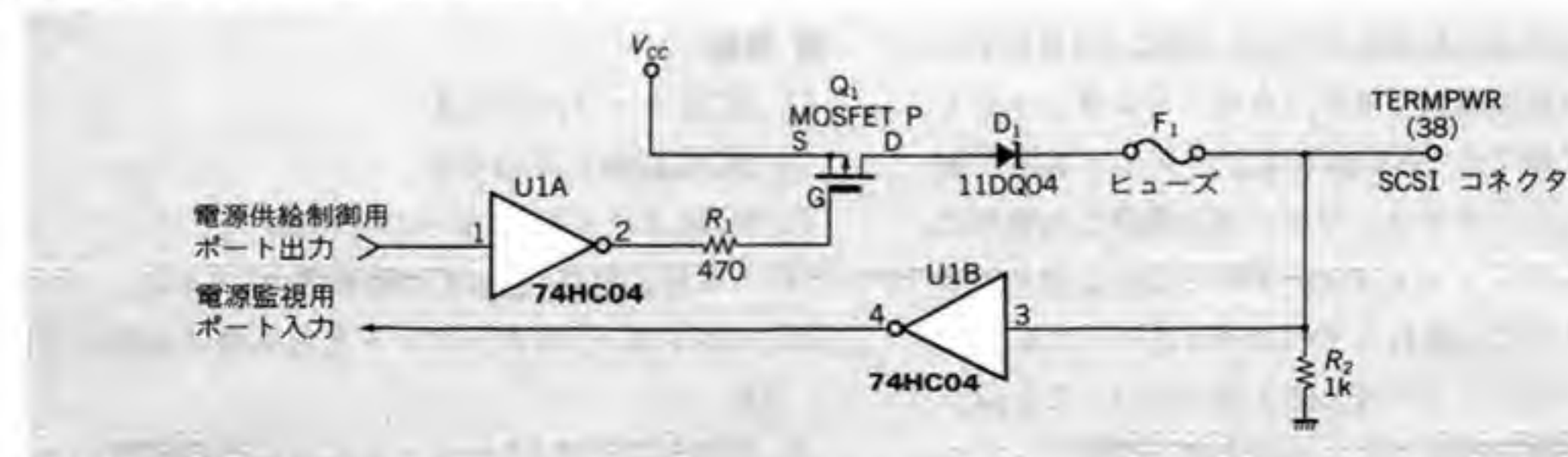


図13 CMOS IC の入力部内部等価回路

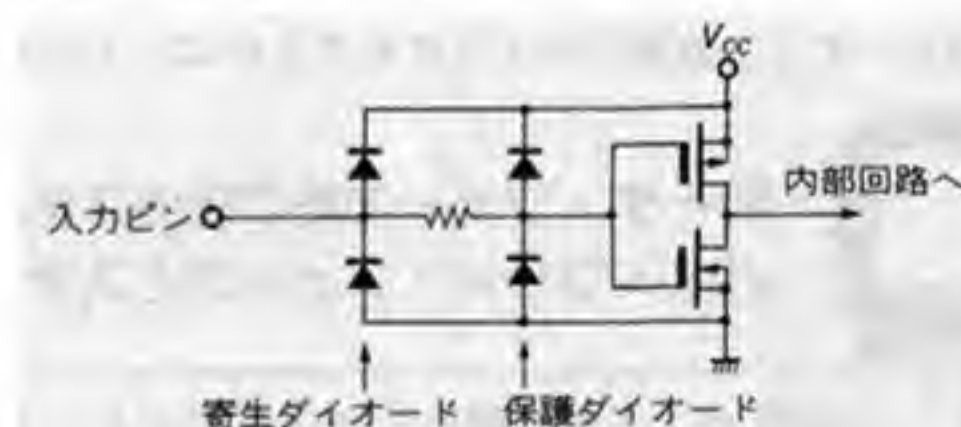


図14 電源 OFF 時の電流経路

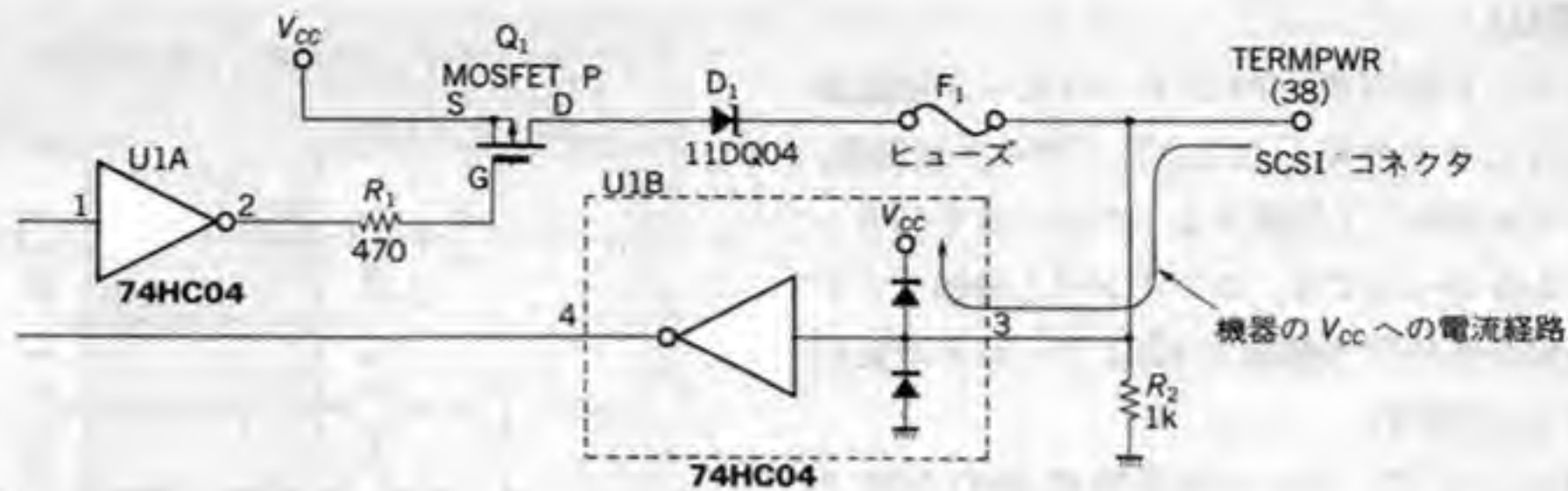


表1 TERMPWR 線への電源供給条件

	SCSI		SCSI-2*	
	供給電圧	電流容量	供給電圧	電流容量
シングルエンド型	4.00~5.25 V DC	800 mA 最小	4.25~5.25 V DC	900 mA 最小
ディファレンシャル型	4.00~5.25 V DC	600 mA 最小	4.00~5.25 V DC	600 mA 最小

* A ケーブルの規定

がありました。この機器の回路の一部分を図12に示します。

このとき、おおかたの動作テストを無事終了した段階で、たまたまこの機器の電源を OFF にして、同じ SCSI バスに接続された他の機器を動作させてみるとうまく転送できないという現象に出会いました。原因は、TERMPWR 線の電源電圧の低下で、監視回路に利用した CMOS IC の内部構造に問題があったのです。

一般に、CMOS IC の入力部内部等価回路は、保護ダイオードと寄生ダイオードによって図13のような構造になっています。このため監視回路に利用した 74HC04 の入力部寄生ダイオードが、TERMPWR 線

からこの機器の V_{cc} への電流経路となり、 V_{cc} への逆電流防止用にあるショットキ・バリア・ダイオードの効果をなくしていたのです。

この機器の電源を OFF にすると、SCSI バスの TERMPWR 線からこの機器の V_{cc} に電流が流れ込んで終端抵抗用電源電圧が極端に下がり、転送動作が不安定となっていました(図14)。TTL 構造の IC を TERMPWR 線の監視回路に利用すれば、このような問題は起こらないのですが、この例では、TTL 構造の IC に余ったゲートがなかったため、74HC04 の入力部前段に 10 k Ω の抵抗を直列に付加するという対策を講じました。

(有吉和久)

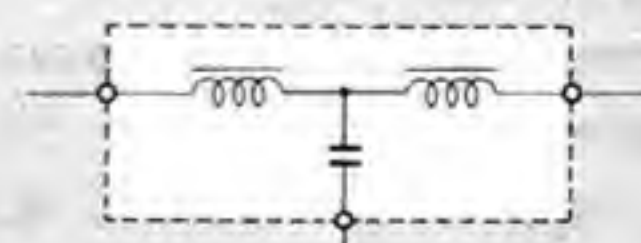
Q EMI フィルタを付加したところ信号グリッジが発生してしまったが?

A 昨今、アメリカの FCC、西ドイツの VDE、そして日本の VCCI など不要輻射ノイズに対する規制が強化され、コンピュータやその周辺装置である SCSI 機器の開発においても、不要輻射ノイズの対策が不可欠になっています。

最近では、コンデンサとビーズ・コアや巻線などのインダクタを組み合わせた各種の小型ノイズ・フィルタ(以下、EMI フィルタ)が、数社の部品メーカーから販売されるようになり、信号線からの輻射ノイズ対策用

として有効に利用できるようになってきました。また、部品メーカーのカatalogには、プリントなどのインター

図16 T 型構成の EMI フィルタ



フェース回路設計時におけるノイズ対策例が紹介されています(図15)。

シングルエンド型 SCSI のインターフェース回路部に、EMI フィルタを利用したために、データ転送時、まれにエラーが発生して問題となったケースを紹介しましょう。このケースでは、コンデンサと巻線インダクタを組み合わせた T 型構成の EMI フィルタが使用されていました(図16)。

オシロスコープで、データ転送時の REQ 信号と ACK 信号を観測してみると、信号のエッジ部でグリッジが発生しており、転送データのハンドシェイク時に何らかの影響を与えているようでした。ためしに、EMI フィルタを外してテストしてみると、先のエラーは解消され、オシロスコープによる観測でも、信号エッジ部のグリッジがなくなっていることが確認できました。

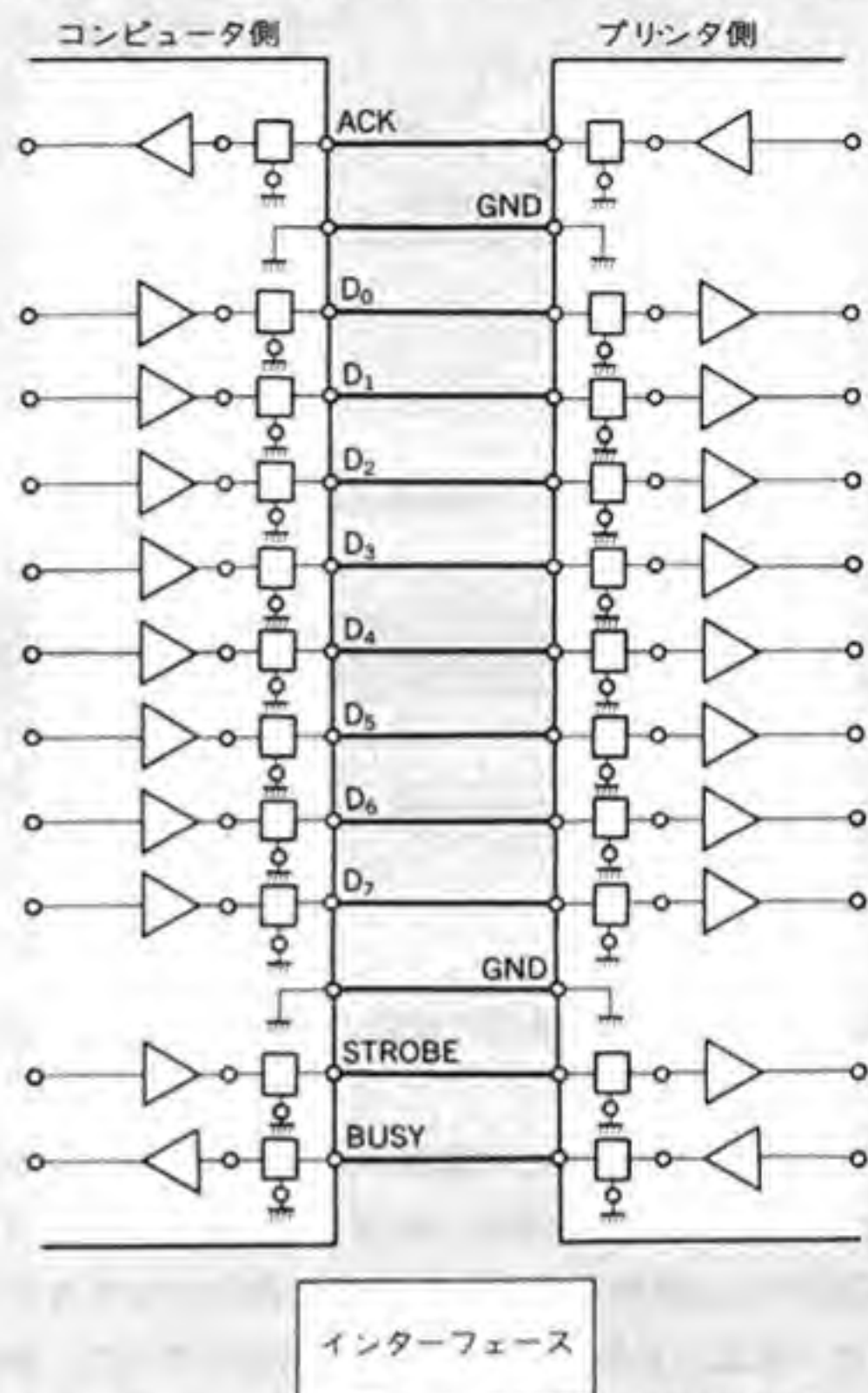
この事例では、転送に影響を与えない程度 of 定数値の EMI フィルタを再選定してもらうことになりましたが、変更した定数値の EMI フィルタで輻射ノイズ・レベルの規制値をクリアできるかどうかは別問題であり、再度の輻射ノイズの確認作業が必要となってしまいました。

一般的な電子機器の開発・評価では、輻射ノイズの確認作業が最終段階となる場合が多くなるため、この段階で EMI フィルタを追加する例をよくみかけますが、この事例のように本来の動作がそれによって損なわれてしまっていることはありません。

SCSI は、18 本の信号線を利用したパラレル・バスであり、しかも立ち上がり/立ち下りの急峻な高速パルス信号を用いたデジタル伝送であるため、広い周波数範囲で不要な輻射ノイズの発生が予想されます。SCSI 機器を開発するときは、設計初期段階から、外装

図15 EMI フィルタの使用例

パソコンとプリンタを接続した場合の輻射ノイズ防止回路例



『TDK 信号ライン用ノイズ・フィルタ・カタログ』より引用

ケースのシールドや基板上の信号パターンの最短化などの輻射ノイズ対策を考慮し、SCSI の伝送系に影響がでる可能性のあるノイズ対策部品を追加しないで済むようにしておくことが望ましいでしょう。

(有吉和久)

Q

情報転送フェーズにおける転送データのパリティ・エラーを検出し、LED を点灯させる監視装置を設計したがうまくいかなかった。なぜか？

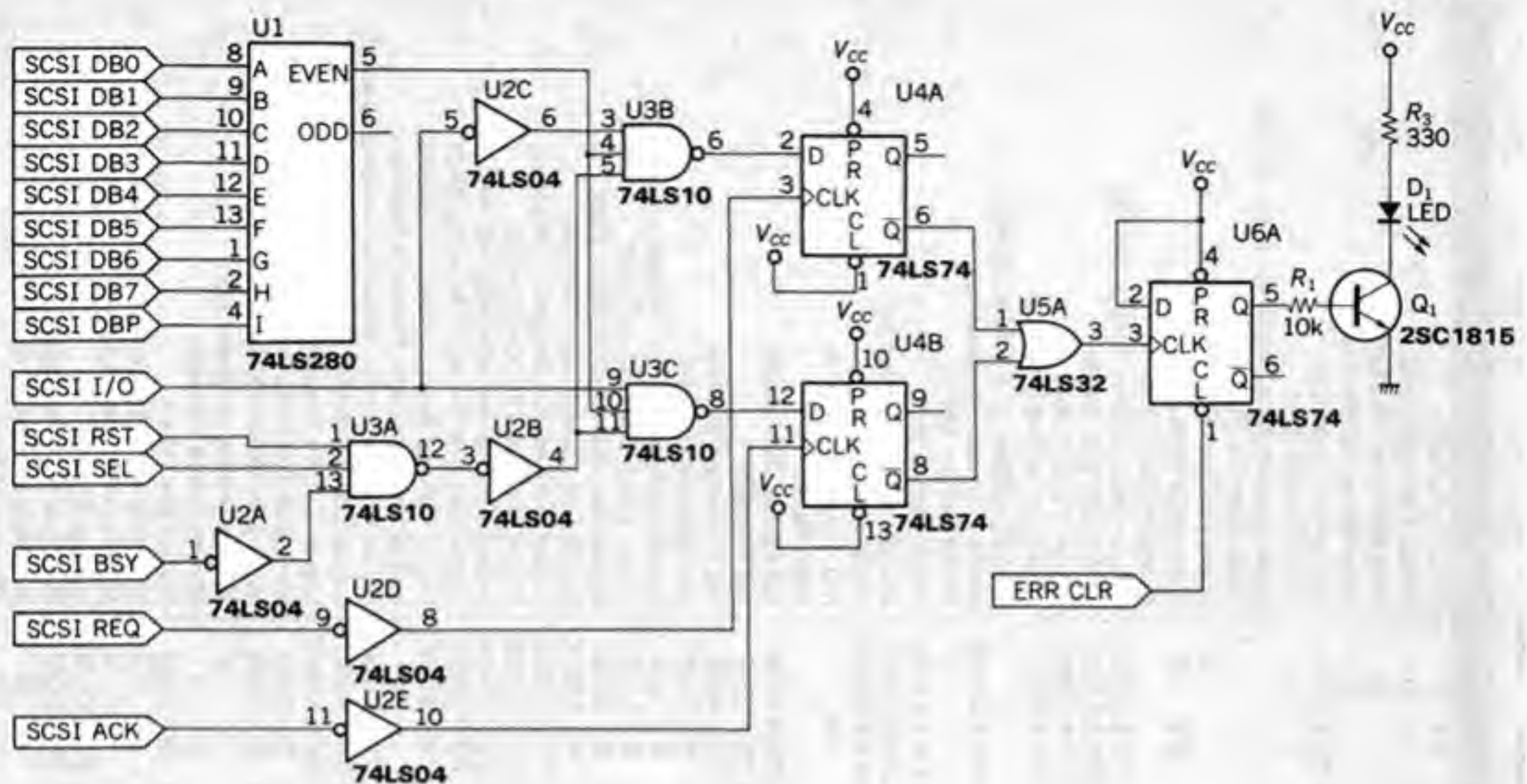
A

SCSI 信号は負論理の考え方をするため、信号の電圧が“L”レベルのときにアクティブ状態で真(“1”)、“H”レベルのときに非アクティブ状態で偽(“0”)と判定されます。この基本的な考え方をよく理解しているつもりでも、SCSI コントロール LSI を利用している分にはとくに意識する必

要がないため、つい忘れがちになり、直接 SCSI 信号を処理する回路を設計したときに思わぬミスをしてしまうことになります。

このケースは、情報転送フェーズにおける転送データのパリティ・エラーを検出し、LED を点灯するように設計した回路の失敗例です(図17)。

図17 問題のある SCSI パリティ・エラー検出回路



SCSI では、正常な転送時のデータ・パリティが奇数となるため、パリティが偶数となったときにエラーと判定すればいいわけです。回路構成は、パリティ・エラーの検出用に標準ロジック IC のメニューにある LS280 (図18) を利用し、データが有効な期間に LS280 の Σ_{Even} が “H” を出力したときにエラーと判定して、その出力をラッチして LED を点灯するものです。

この回路の問題点は、LS280 の入力 A~I にデータ信号 (DB0~DB7) とパリティ信号 (DBP) が接続されており、この 9 本の信号のうち偶数本の信号が “H” レベルとなったときにエラーと判定したことに起因しています。SCSI でいう正常転送時のデータ・パリティが奇数とは、論理 1 のデータ信号 (DB0~DB7) とパリティ信号 (DBP) の数が正常時には奇数本あるということであり、換言すれば、この 9 本の信号のうち奇数本の信号が “L” レベルとなることを意味しています。

したがって、エラーの判定には、この 9 本の信号のうち偶数本の信号が “L” レベルである、すなわち奇数本の信号が “H” レベルとなったときにエラーと判定しなければなりません。

原因がわかれば対策は簡単です。LS280 の Σ_{Even} の代わりに Σ_{Odd} の出力を利用すればよいことが真理値表 (表 2) からわかります。

(有吉和久)

図18 74LS280 のピン配置

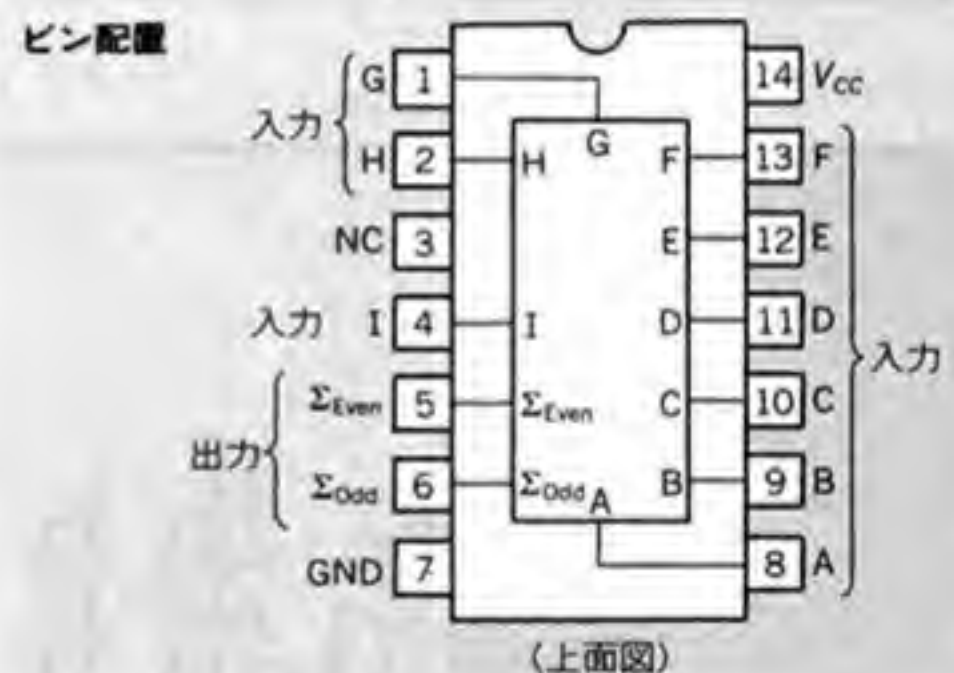


表 2 74LS280 の真理値表

入力 A~I の “H” の数	出力	
	Σ_{Even}	Σ_{Odd}
0, 2, 4, 6, 8	“H”	“L”
1, 3, 5, 7, 9	“L”	“H”

ありよし・かずひさ 積水化学工業株式会社 電子研究所 メカトロ機器開発センター

しみず・てつお 株式会社 東陽テクニカ エレクトロニクス事業部 開発部

まき・よしとも SCSI 機器開発技術者

よしだ・ひろゆき 富士通デバイス株式会社 技術本部 第三設計部

		D:ダイレクト・アクセス・デバイス T:シーケンシャル・アクセス・デバイス L:プリンタ・デバイス P:プロセッサ・デバイス W:ライト・ワンス・デバイス R:CD-ROMデバイス S:スキャナ・デバイス O:光メモリ・デバイス M:メディア・チェンジャ・デバイス C:コミュニケーション・デバイス		エラーの内容	
ASC	ASCQ	DTLPWRSOMC			
13h	00h	D W O	Address Mark Not Found For Data Field		
12h	00h	D W O	Address Mark Not Found For ID Field		
00h	11h	R	Audio Play Operation In Progress		
00h	12h	R	Audio Play Operation Paused		
00h	14h	R	Audio Play Operation Stopped Due To Error		
00h	13h	R S	Audio Play Operation Successfully Completed		
00h	04h	T	Beginning-Of-Partition/Medium Detected		
14h	04h	T	Block Sequence Error		
30h	02h	DT WR O	Cannot Read Medium-Incompatible Format		
30h	01h	DT WR O	Cannot Read Medium-Unknown Format		
52h	00h	T	Cartridge Fault		
3Fh	02h	DTLPWRSOMC	Changed Operating Definition		
11h	06h	WR O	Circ Unrecovered Error		
30h	03h	DT	Cleaning Cartridge Installed		
4Ah	00h	DTLPWRSOMC	Command Phase Error		
2Ch	00h	DTLPWRSOMC	Command Sequence Error		
2Fh	00h	DTLPWRSOMC	Commands Cleared By Another Initiator		
2Bh	00h	DTLPWRSO C	Copy Cannot Execute Since Host Cannot Disconnect		
41h	00h	D	Data Path Failure (Should Use 40 NN)		
48h	00h	DTLPWRSOMC	Data Phase Error		
11h	07h	W O	Data Resynchronization Error		
16h	00h	W O	Data Synchronization Mark Error		
19h	00h	D	Defect List Error		
19h	03h	D	Defect List Error In Grown List		
19h	02h	D	Defect List Error In Primary List		
19h	01h	D	Defect List Not Available		
1Ch	00h	D	Defect List Not Found		
32h	01h	D W O	Defect List Update Failure		
40h	NNh	DTLPWRSOMC	Diagnostic Failure On Component NN (80h~FFh)		
63h	00h	R S	End Of User Area Encountered On This Track		
00h	05h	T	End-Of-Data Detected		
14h	03h	T	End-Of-Data Not Found		
00h	02h	T S	End-Of-Partition/Medium Detected		
51h	00h	T	Erase Failure		
0Ah	00h	DTLPWRSOMC	Error Log Overflow		
11h	02h	DT W SO	Error Too Long To Correct		
03h	02h	T	Excessive Write Errors		
33h	07h	L	Failed To Sense Bottom-Of-Form		
3Bh	06h	L	Failed To Sense Top-Of-Form		
00h	01h	T	Filemark Detected		
14h	02h	T	Filemark Or Setmark Not Found		

ASC	ASCQ	DTLPWRSOMC	エラーの内容
09h	02h	WR O	Focus Servo Failure
31h	01h	D L O	Format Command Failed
58h	00h	O	Generation Does Not Exist
1Ch	02h	D	Grown Defect List Not Found
00h	06h	DTLPWRSOMC	I/O Process Terminated
10h	00h	W O	ID CRC Or ECC Error
22h	00h	D	Illegal Function(Should Use 20 00, 24 00, Or 26 00)
64h	00h	R	Illegal Mode For This Track
28h	01h	M	Import Or Export Element Accessed
30h	00h	DT WR OM	Incompatible Medium Installed
11h	08h	T	Incomplete Block Read
48h	00h	DTLPWRSOMC	Initiator Detected Error Message Received
3Fh	03h	DTLPWRSOMC	Inquiry Data Has Changed
44h	00h	DTLPWRSOMC	Internal Target Failure
3Dh	00h	DTLPWRSOMC	Invalid Bits In Identify Message
2Ch	02h	S	Invalid Combination Of Windows Specified
20h	00h	DTLPWRSOMC	Invalid Command Operation Code
21h	01h	M	Invalid Element Address
24h	00h	DTLPWRSOMC	Invalid Field In CDB
26h	00h	DTLPWRSOMC	Invalid Field In Parameter List
49h	00h	DTLPWRSOMC	Invalid Message Error
11h	05h	WR O	L-EC Uncorrectable Error
60h	00h	S	Lamp Failure
5Bh	02h	DTLPWRSOM	Log Counter At Maximum
5Bh	00h	DTLPWRSOM	Log Exception
5Bh	03h	DTLPWRSOM	Log List Codes Exhausted
2Ah	02h	DTLPWRSOMC	Log Parameters Changed
21h	00h	DT WR OM	Logical Block Address Out Of Range
08h	00h	DTLPWRSOMC	Logical Unit Communication Failure
08h	02h	DTLPWRSOMC	Logical Unit Communication Parity Error
08h	01h	DTLPWRSOMC	Logical Unit Communication Time-Out
05h	00h	DTLPWRSOMC	Logical Unit Does Not Respond To Selection
4Ch	00h	DTLPWRSOMC	Logical Unit Failed Self-Configuration
3Eh	00h	DTLPWRSOMC	Logical Unit Has Not Self-Configured Yet
04h	01h	DTLPWRSOMC	Logical Unit Is In Process Of Becoming Ready
04h	00h	DTLPWRSOMC	Logical Unit Not Ready, Cause Not Reportable
04h	00h	DTLPWRSOMC	Logical Unit Not Ready, Format In Progress
04h	04h	DTLPWRSOMC	Logical Unit Not Ready, Initializing Command Required
04h	02h	DTLPWRSOMC	Logical Unit Not Ready, Manual Intervention Required
04h	03h	DTLPWRSOMC	Logical Unit Not Supported
25h	00h	DTLPWRSOMC	Mechanical Positioning Error
15h	01h	DTLPWRSOM	Media Load Or Eject Failed
53h	00h	DTLPWRSOM	Medium Destination Element Full
3Bh	0Dh	M	Medium Format Corrupted
31h	00h	DT W O	Medium Not Present
3Ah	00h	DTLPWRSOM	Medium Removal Reviented
53h	02h	DT WR OM	Medium Source Element Empty
3Bh	0Eh	M	Message Error
43h	00h	DTLPWRSOMC	Microcode Has Been Changed
3Fh	01h	DTLPWRSOMC	Miscompare During Verify Operation
1Dh	00h	D W O	Miscorrected Error
11h	0Ah	DT	Mode Parameters Changed
2Ah	01h	DTLPWRSOMC	

ASC	ASCQ	DTL PWR S OMC	エラーの内容
07h	00h	DTL WRS OMC	Multiple Peripheral Devices Selected
11h	03h	DT W S O	Multiple Read Errors
00h	00h	DTL PWR S OMC	No Additional Sense Information
00h	15h	R	No Current Audio Status To Return
32h	00h	D W O	No Defect Spare Location Available
11h	09h	T	No Gap Found
01h	00h	D W O	No Index/Sector Signal
06h	00h	D WR OMC	No Reference Position Found
02h	00h	D WR OMC	No Seek Complete
03h	01h	T	No Write Current
28h	00h	DTL PWR S OMC	Not Ready To Ready Transition, Medium May Have Changed
5Ah	01h	DT WR OMC	Operator Medium Removal Request
5Ah	00h	DTL PWR S OMC	Operator Request Or State Change Input (Unspecified)
5Ah	03h	DT W O	Operator Selected Write Permit
5Ah	02h	DT W O	Operator Selected Write Protect
61h	02h	S	Out Of Focus
4Eh	00h	DTL PWR S OMC	Overlapped Commands Attempted
2Dh	00h	T	Overwrite Error On Update In Place
3Bh	05h	L	Paper Jam
1Ah	00h	DTL PWR S OMC	Parameter List Length Error
26h	01h	DTL PWR S OMC	Parameter Not Supported
26h	02h	DTL PWR S OMC	Parameter Value Invalid
2Ah	00h	DTL WRS OMC	Parameters Changed
03h	00h	DTL W S O	Peripheral Device Write Fault
50h	02h	T	Position Error Related To Timing
3Bh	0Ch	S	Position Past Beginning Of Medium
3Bh	0Bh	S	Position Past End Of Medium
15h	02h	DT WR O	Positioning Error Detected By Read Of Medium
29h	00h	DTL PWR S OMC	Power On, Reset, Or Bus Device Reset Occurred
42h	00h	D	Power-on Or Self-Test Failure (Should Use 40 NN)
1Ch	01h	D O	Primary Defect List Not Found
40h	00h	D	RAM Failure (Should Use 40 NN)
15h	00h	DTL WRS OMC	Random Positioning Error
3Bh	0Ah	S	Read Past Beginning Of Medium
3Bh	09h	S	Read Past End Of Medium
11h	01h	DT W S O	Read Retries Exhausted
14h	01h	DT WR O	Record Not Found
14h	00h	DTL WRS O	Recorded Entity Not Found
18h	02h	D WR O	Recovered Data-Data Auto Reallocated
18h	05h	D WR O	Recovered Data-Recommend Reassignment
18h	06h	D WR O	Recovered Data-Recommend Rewrite
17h	05h	D WR O	Recovered Data Using Previous Sector ID
18h	03h	R	Recovered Data With CIRC
18h	01h	D WR O	Recovered Data With Error Correction & Retries Applied
18h	00h	D WR O	Recovered Data With Error Correction Applied
18h	04h	R	Recovered Data With L-EC
17h	03h	DT WR O	Recovered Data With Negative Head Offset
17h	00h	DT WR S O	Recovered Data With No Error Correction Applied
17h	02h	DT WR O	Recovered Data With Positive Head Offset
17h	01h	DT WRS O	Recovered Data With Retries
17h	04h	WR O	Recovered Data With Retries AND/OR CIRC Applied
17h	06h	D W O	Recovered Data Without ECC-Data Auto-Reallocated

ASC	ASCQ	DTL PWR S OMC	エラーの内容
17h	07h	D W O	Recovered Data Without ECC-Recommend Reassignment
17h	08h	D W O	Recovered Data Without ECC-Recommend Rewrite
1Eh	00h	D W O	Recovered ID With ECC Correction
3Bh	08h	T	Reposition Error
36h	00h	L	Ribbon, Ink, Or Toner Failure
37h	00h	DTL WRS OMC	Rounded Parameter
5Ch	00h	D O	RPL Status Change
39h	00h	DTL WRS OMC	Saving Parameters Not Supported
62h	00h	S	Scan Head Positioning Error
47h	00h	DTL PWR S OMC	SCSI Parity Error
54h	00h	P	SCSI To Host System Interface Failure
45h	00h	DTL PWR S OMC	Select Or Reselect Failure
3Bh	00h	TL	Sequential Positioning Error
00h	03h	T	Setmark Detected
3Bh	04h	L	Slew Failure
09h	03h	WR O	Spindle Servo Failure
5Ch	02h	D O	Spindles Not Synchronized
5Ch	01h	D O	Spindles Synchronized
1Bh	00h	DTL PWR S OMC	Synchronous Data Transfer Error
55h	00h	P	System Resource Failure
33h	00h	T	Tape Length Error
3Bh	03h	L	Tape Or Electronic Vertical Forms Unit Not Ready
3Bh	01h	T	Tape Position Error At Beginning-Of-Medium
3Bh	02h	T	Tape Position Error At End-Of-Medium
3Fh	00h	DTL PWR S OMC	Target Operating Conditions Have Changed
5Bh	01h	DTL PWR S OMC	Threshold Condition Met
26h	03h	DTL PWR S OMC	Threshold Parameters Not Supported
2Ch	01h	S	Too Many Windows Specified
09h	00h	DT WR O	Track Following Error
09h	01h	WR O	Tracking Servo Failure
61h	01h	S	Unable To Acquire Video
57h	00h	R	Unable To Recover Table-Of-Contents
53h	01h	T	Unload Tape Failure
11h	00h	DT WRS O	Unrecovered Read Error
11h	04h	D W O	Unrecovered Read Error-Auto Reallocate Failed
11h	0Bh	D W O	Unrecovered Read Error-Recommend Reassignment
11h	0Ch	D W O	Unrecovered Read Error-Recommend Rewrite The Data
46h	00h	DTL PWR S OMC	Unsuccessful Soft Reset
59h	00h	O	Updated Block Read
61h	00h	S	Video Acquisition Error
50h	00h	T	Write Append Error
50h	01h	T	Write Append Position Error
0Ch	00h	T S	Write Error
0Ch	02h	D W O	Write Error-Auto Reallocation Failed
0Ch	01h	D W O	Write Error Recovered With Auto Reallocation
27h	00h	DT W O	Write Protected
80h	XXh	}	ベンダ固有 ASC/ASCQ
FFh	XXh	}	ベンダ固有 ASCQ (YYh は標準の ASC)
YYh	80h	}	
YYh	FFh	}	

注) この表に定義されないコードはリザーブとする。

規格化進む SCSI-3の概要

Pケーブル/Qケーブル/16ビット幅転送/32ビット幅転送/SCSI ID拡張/
コマンド拡張/メッセージ拡張

菅谷 誠一

SCSI-3は、SCSI-2との互換性が維持されるパラレルと、光ファイバなどでの応用が意識されたシリアルに大別できる。すでに「SCSI-3対応」をうたう製品もでてきているが、この規格はまだ最終的に決まったわけではない。SCSI-3では、1本で16ビット幅のWIDE転送が可能なPケーブルとPおよびQケーブルで32ビット幅WIDE転送に対応する方法が導入された。PケーブルはSCSI-2のAケーブルとの混在接続ができる。SCSI IDも、Pケーブルで最大16、P/Qケーブルで最大32に拡張された…といったように、ここではSCSI-2と比較しながら、SCSI-3の概要を紹介する。また、SCSI-3になって用意された二つのメッセージ、Continue I/O ProcessとTarget Transfer Disableの典型的な使用例を示す。

(編集部)

現在ANSIでは、次世代のSCSI規格としてSCSI-3の標準化作業が急ピッチで進められています。SCSI-3では、物理層として、現在のSCSI-2との接続互換性を維持した“パラレル・インターフェース”と、ファイバ・チャネルなどの高速かつ遠距離接続が可能な新しいテクノロジーへの展開をはかる“シリアル・インターフェース”の2種類が規格化される予定です。

そして、プロトコルとしては、SCSI-2の方式を踏襲した“インターロックド・プロトコル”と、シリアル・インターフェースへの対応や転送の効率化をはかるための“パケットイズド・プロトコル”の2種類があります。前者の“インターロックド・プロトコル”はパラレル・インターフェース上で使用され、SCSI-2との下位互換性が維持される予定です。

SCSI-3パラレル・インターフェースでは、物理レベルの条件(コネクタ、ケーブルなど)がSCSI-2の規定

から変更され、とくにWIDE SCSIの組み込み仕様が大きく異なっています。そして、すでにこのSCSI-3の規定を適用したWIDE SCSIの製品が登場しはじめています。

ここでは、1993年8月までに明らかになっているSCSI-3パラレル・インターフェース(SPI: SCSI-3 Parallel Interface)とインター・ロックド・プロトコル(SIP: SCSI-3 Interlocked Protocol)のなかから、WIDE SCSI(16/32ビットSCSI)とメッセージ・システムの拡張に関するおもな内容を紹介します。すでに製品化が開始されているとはいえ、ここに記述する内容は最終的なSCSI規格にはなっていないことに留意してください。

1

インターフェース規定の概要

SCSI-3パラレル・インターフェースの最大の特徴は、SCSI-2でのA/Bケーブルの規定を削除し、あらたに1本のケーブルで16ビット幅までのWIDE SCSIが可能なPケーブルと、オプションで32ビット幅のWIDE SCSIを適用するためのQケーブルを導入したことにあります。そして、データ・バス幅の拡張にあわせて、最大16台あるいは32台までのSCSIデバイスを、1本のバス上に接続できるようにしています。

以下に、SCSI-2から変更される予定になっているおもな規定とその概要を述べます。

■ P/Q ケーブル

SCSI-2のAケーブルとWIDE SCSIを実現するためのBケーブル(オプション)の規定は廃止され、あら

たに P ケーブルと Q ケーブルが規定されます。いずれも 68 信号線の構成となります。そして、コネクタは、P ケーブルおよび Q ケーブルとも、68 ピン、50 mil ピッチの高密度コネクタとなります。

P ケーブルには 16 ビットのデータ・バスが定義されていますので、P ケーブル 1 本だけで、16 ビット幅の WIDE 転送ができるようになりました。

SCSI-2 では SCSI-1 との物理的接続互換性を重視したため、基本構成を 8 ビット幅データ・バスの A ケーブルとし、WIDE SCSI を適用するためにはオプションの B ケーブルを必要とする構成としました。しかし、装置の小型化と高性能化が急激に進展し、たとえば、3.5 インチ・サイズの磁気ディスク装置でも 16 ビット転送の必要性が増大してくる状況にあつては、A/B ケーブルの規定が時代の要請にそぐわなくなっています。このことが P ケーブルを導入した最大の理由といえるでしょう。

Q ケーブルには、追加の 16 ビット幅のデータ・バスが定義されます。そして、32 ビット幅の WIDE 転送を行うときには、P/Q ケーブルの二つのコネクタを備えることになります。規格化を討議している過程では、32 ビット幅の WIDE SCSI も 1 本のケーブルで実現できるようなオプションも検討されましたが、P/Q ケーブルの規定だけに統合される予定となっています。

■ SCSI-2 との混在接続

混在接続用のアダプタを用意することによって、P ケーブルは SCSI-2 の A ケーブルと接続することができます。SCSI-2 を正しく適用した SCSI デバイスは、SCSI-3 で追加されたプロトコルをリジェクトし、そして SCSI-3 を適用した SCSI デバイスは SCSI-2 で規定されている範囲内のプロトコルで動作することができますので、混在接続された両者の SCSI デバイスは同一の SCSI バス上で、正しく動作することが可能です。

■ SCSI ID の拡張

SCSI-2 での WIDE SCSI の規定は、データ転送、つまりデータ・イン・フェーズとデータ・アウト・フェーズだけに限定されていました。SCSI-3 で導入される P/Q ケーブルでは、拡張されたデータ・バスのすべてのビット対応に SCSI ID を割り付けることを可能としています。

つまり、P ケーブルでは最大 16 台まで、P/Q ケーブルでは最大 32 台までの SCSI デバイスを一つの SCSI バスに接続することができます。

インターフェース・ケーブルの最大長の規定は SCSI-2 と同一(シングルエンド型で 6 m、ディファレンシャル型で 25 m)となっていますが、システムや入出力機器の小型化にともない、有意義な機能拡張であるといえます。しかし、SCSI デバイスの接続台数の増加により伝送系としての特性はより厳しくなりますので、バス上の信号品質には十分に注意する必要があります。

プロトコル規定での SCSI ID の拡張は、具体的には、アービトレーション・フェーズとセレクション・フェーズおよびリセレクション・フェーズで実現されています。そして、SCSI-2 の A ケーブル、つまり 8 ビット SCSI の機器との接続互換をとるために、アービトレーション・フェーズでのプライオリティ定義やセレクション/リセレクション・フェーズでのデータ・バスのパリティ・チェックの実施方法に工夫がこらされています。

■ コマンド定義の拡張

SCSI ID の拡張により、CDB やパラメータ・リスト上にイニシエータやターゲットの SCSI ID を指定するフィールドが定義されているいくつかのコマンドが影響を受けます。

代表的なものとしては、Reserve コマンドと Release コマンドの“サード・パーティ・リザーブ/リリース機能”や、Copy コマンドで相手先の SCSI デバイスやデータ属性を指定するためのセグメント・ディスクリプタ・パラメータなどがあります。

SCSI-3 の論理仕様はまだ明らかになっていませんが、10 バイト長の Reserve、Release コマンドを追加したり、あらたなセグメント・ディスクリプタ・パラメータを追加することにより、これらのコマンドへの拡張機能の導入をはかることになっています。

2

物理規定

P/Q ケーブルの導入による物理的な規定は以下のようになっています。伝送系には、SCSI-2 と同様にシ

シングルエンド型とディファレンシャル型の2種類が規定されます。

■ P/Q ケーブルとコネクタ

SCSI デバイスは、68 信号構成の P ケーブルで相互にデジィ・チェーン接続されます。また、32 ビット幅の WIDE 転送機能を備える SCSI デバイス間には、68 信号構成の Q ケーブルを追加接続します。

P ケーブルと Q ケーブル用のコネクタは、68 ピン構成、50 mil ピッチ 2 列の高密度型コネクタとなります。SCSI-2 と同様にシールド型とノン・シールド型の2種

類が規定されます。コネクタのコンタクト機構や勘合の物理的仕様は SCSI-2 の B ケーブルと同一のピン・コンタクト型ですが、コネクタの抜け防止機構がラッチ型からスクリュウ・ロック型に変更される予定です。

表1～表4に、P ケーブルと Q ケーブルのシングルエンド型とディファレンシャル型でのコネクタ上の信号割り付けを示します。

Q ケーブルには、追加のデータ・バス専用の転送タイミング信号として、REQQ 信号と ACKQ 信号が設けられています。したがって、16 ビット・バスの SCSI デバイスと 32 ビット・バスの SCSI デバイスとを一つの

表1 シングルエンド型バス-P ケーブル信号割り付け

コンタクト 番号	信 号	信 号	コンタクト 番号
01	GND	-DB(12)	35
02	GND	-DB(13)	36
03	GND	-DB(14)	37
04	GND	-DB(15)	38
05	GND	-DB(P1)	39
06	GND	-DB(0)	40
07	GND	-DB(1)	41
08	GND	-DB(2)	42
09	GND	-DB(3)	43
10	GND	-DB(4)	44
11	GND	-DB(5)	45
12	GND	-DB(6)	46
13	GND	-DB(7)	47
14	GND	-DB(P)	48
15	GND	GND	49
16	GND	GND	50
17	TERMPWR	TERMPWR	51
18	TERMPWR	TERMPWR	52
19	(リザーブ)	(リザーブ)	53
20	GND	GND	54
21	GND	-ATN	55
22	GND	GND	56
23	GND	-BSY	57
24	GND	-ACK	58
25	GND	-RST	59
26	GND	-MSG	60
27	GND	-SEL	61
28	GND	-C/D	62
29	GND	-REQ	63
30	GND	-I/O	64
31	GND	-DB(8)	65
32	GND	-DB(9)	66
33	GND	-DB(10)	67
34	GND	-DB(11)	68

表2 シングルエンド型バス-Q ケーブル信号割り付け

コンタクト 番号	信 号	信 号	コンタクト 番号
01	GND	-DB(28)	35
02	GND	-DB(29)	36
03	GND	-DB(30)	37
04	GND	-DB(31)	38
05	GND	-DB(P3)	39
06	GND	-DB(16)	40
07	GND	-DB(17)	41
08	GND	-DB(18)	42
09	GND	-DB(19)	43
10	GND	-DB(20)	44
11	GND	-DB(21)	45
12	GND	-DB(22)	46
13	GND	-DB(23)	47
14	GND	-DB(P2)	48
15	GND	GND	49
16	GND	GND	50
17	TERMPWRQ	TERMPWRQ	51
18	TERMPWRQ	TERMPWRQ	52
19	(リザーブ)	(リザーブ)	53
20	GND	GND	54
21	GND	(Terminated)	55
22	GND	GND	56
23	GND	(Terminated)	57
24	GND	-ACKQ	58
25	GND	(Terminated)	59
26	GND	(Terminated)	60
27	GND	(Terminated)	61
28	GND	(Terminated)	62
29	GND	-REQQ	63
30	GND	(Terminated)	64
31	GND	-DB(24)	65
32	GND	-DB(25)	66
33	GND	-DB(26)	67
34	GND	-DB(27)	68

SCSI バスに混在接続するときの考えかたや方法は、SCSI-2 の A/B ケーブルの場合と同一です。

Q ケーブル上の未使用ピンで、P ケーブルでは信号が割り当てられている位置は、P ケーブルと Q ケーブルの誤接続による破損を防止するため、他の一般信号と同様にターミネータ回路によって終端処理をするように規定されています(表 2, 表 4 で(Terminated)と指定されている位置)。

なお、シングルエンド型の伝送系で FAST SCSI(高速同期モード転送)を実施するときは、インターフェース・ケーブルの最大長を 3m に制限することが“推奨”

されています。

■ A ケーブルとの混在接続

P ケーブルと SCSI-2 の A ケーブルとを接続する場合は、一般には変換アダプタ回路が必要になると考えられます。

図 1 に変換アダプタ回路の構成例と SCSI バスの接続形態例を示します。データ・バス幅が広い P ケーブル上の DB15~8 と DBP1 信号は、このアダプタ回路上で終端されます。また、データ・バス幅が狭い A ケーブルの SCSI デバイスがバスを駆動するときは、当

表 3 デイファレンシャル型バス-P ケーブル信号割り付け

コンタクト 番号	信 号	信 号	コンタクト 番号
01	+DB(12)	-DB(12)	35
02	+DB(13)	-DB(13)	36
03	+DB(14)	-DB(14)	37
04	+DB(15)	-DB(15)	38
05	+DB(P1)	-DB(P1)	39
06	GND	GND	40
07	+DB(0)	-DB(0)	41
08	+DB(1)	-DB(1)	42
09	+DB(2)	-DB(2)	43
10	+DB(3)	-DB(3)	44
11	+DB(4)	-DB(4)	45
12	+DB(5)	-DB(5)	46
13	+DB(6)	-DB(6)	47
14	+DB(7)	-DB(7)	48
15	+DB(P)	-DB(P)	49
16	DIFFSENS	GND	50
17	TERMPWR	TERMPWR	51
18	TERMPWR	TERMPWR	52
19	(リザーブ)	(リザーブ)	53
20	+ATN	-ATN	54
21	GND	GND	55
22	+BSY	-BSY	56
23	+ACK	-ACK	57
24	+RST	-RST	58
25	+MSG	-MSG	59
26	+SEL	-SEL	60
27	+C/D	-C/D	61
28	+REQ	-REQ	62
29	+I/O	-I/O	63
30	GND	GND	64
31	+DB(8)	-DB(8)	65
32	+DB(9)	-DB(9)	66
33	+DB(10)	-DB(10)	67
34	+DB(11)	-DB(11)	68

表 4 デイファレンシャル型バス-Q ケーブル信号割り付け

コンタクト 番号	信 号	信 号	コンタクト 番号
01	+DB(28)	-DB(28)	35
02	+DB(29)	-DB(29)	36
03	+DB(30)	-DB(30)	37
04	+DB(31)	-DB(31)	38
05	+DB(P3)	-DB(P3)	39
06	GND	GND	40
07	+DB(16)	-DB(16)	41
08	+DB(17)	-DB(17)	42
09	+DB(18)	-DB(18)	43
10	+DB(19)	-DB(19)	44
11	+DB(20)	-DB(20)	45
12	+DB(21)	-DB(21)	46
13	+DB(22)	-DB(22)	47
14	+DB(23)	-DB(23)	48
15	+DB(P2)	-DB(P2)	49
16	DIFFSENS	GND	50
17	TERMPWRQ	TERMPWRQ	51
18	TERMPWRQ	TERMPWRQ	52
19	(リザーブ)	(リザーブ)	53
20	(Terminated)	(Terminated)	54
21	GND	GND	55
22	(Terminated)	(Terminated)	56
23	+ACKQ	-ACKQ	57
24	(Terminated)	(Terminated)	58
25	(Terminated)	(Terminated)	59
26	(Terminated)	(Terminated)	60
27	(Terminated)	(Terminated)	61
28	+REQQ	-REQQ	62
29	(Terminated)	(Terminated)	63
30	GND	GND	64
31	+DB(24)	-DB(24)	65
32	+DB(25)	-DB(25)	66
33	+DB(26)	-DB(26)	67
34	+DB(27)	-DB(27)	68

然 DB15~8 と DBP1 信号はパッシブな状態となり、ターミネータ回路によって信号の状態は False (パリティがインバリッドな状態) となっています。

なお、P ケーブルをサポートする SCSI デバイスでターミネータ回路を装置内部に実装している場合は、DB15~8, P1 とそれ以外の信号とにターミネータ回路を分解しておくことによって、変換ケーブルを用意するだけで、A ケーブルとの接続ができるようになります。

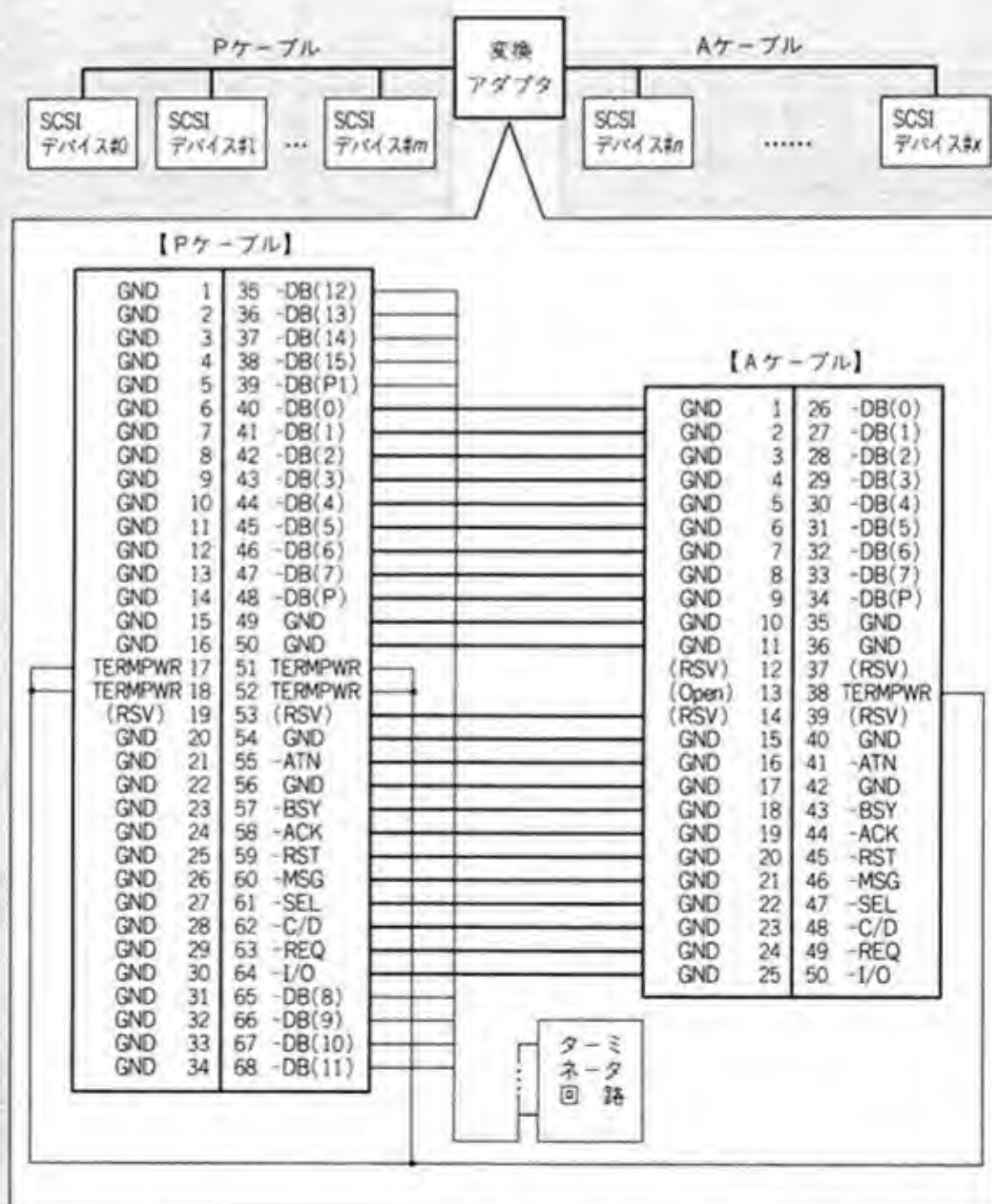
■ 電気的条件

シングルエンド型およびディファレンシャル型とも、基本的な要件は SCSI-2 での規定と同様と考えてさし

つかえありませんが、伝送特性と信号品質の向上をはかるため、ケーブルの特性インピーダンスや伝播遅延時間、レシーバ回路とドライバ回路の入出力特性などに、より詳細な要求特性が追加される予定です。

とくにシングルエンド型の伝送系で高速同期モード転送を行う場合については、信号の状態が“False”のときでもドライバ回路がその信号を駆動する“アクティブ・ネゲーション”型を、REQ/ACK 信号とデータ・バスに導入することが“推奨”として規定される予定です。この方法は、FAST SCSI の厳しいタイミング規定を満たすためにはきわめて有利であり、実際に、最近発表されているプロトコル LSI には、オン・チップのアクティブ・ネゲーション型ドライバを備えたも

図1 PケーブルとAケーブルの接続



のが多くなっています。

3

プロトコル規定

SCSI ID の拡張を達成するため、プロトコルの規定に若干の機能追加が行われます。具体的には、

- ① 拡張されたデータ・バスを含んで全ビット位置への SCSI アドレス (SCSI ID) の割り付け
- ② アービトレーション・フェーズでのバス使用权の優先順位の定義
- ③ セレクション・フェーズおよびリセレクション・フェーズでの応答方法、とくにデータ・バスのパリティ・チェック方法の規定

があります。

これらの追加機能は、SCSI-2 を適用している SCSI デバイスとの共存ができるように工夫されています。

■ SCSI ID の拡張

データ・バスのすべてのビット位置に、そのビット番号に対応した SCSI ID が割り当てられます。したがって、P ケーブルだけのシステムでは ID#0～#15 まで最大 16 台の SCSI デバイスが、P/Q ケーブルのシステムでは ID#0～#31 まで最大 32 台の SCSI デバイスが存在できます。

■ アービトレーションの優先順位

SCSI アドレスが最大 32 台まで拡張されたので、それに対応してアービトレーション・フェーズでのバス使用权の優先順位を定義し直す必要があります。

表 5 に A ケーブル、P ケーブルおよび P/Q ケーブルでのアービトレーションの優先順位の定義を示します。表からわかるように、P ケーブルで拡張されたデータ・バス・ビット (DB8～15) に対応する SCSI アドレスのアービトレーション優先順位は、A ケーブルと混在接続したときも動作できるように考慮されています。同様に、Q ケーブル上の SCSI アドレスの優先度は、P ケーブルと P/Q ケーブルの混在接続が可能ないように割り当てられています。

■ セレクション/リセレクション・フェーズ

セレクション・フェーズおよびリセレクション・フ

表 5 アービトレーションの優先順位 (16/32 ビット SCSI)

アービトレーション 優先順位	A ケーブル	P ケーブル	P/Q ケーブル
(最高位) 1	SCSI ID=7	SCSI ID=7	SCSI ID=7
2	6	6	6
3	5	5	5
4	4	4	4
5	3	3	3
6	2	2	2
7	1	1	1
8	0	0	0
9	—	15	15
10	—	14	14
11	—	13	13
12	—	12	12
13	—	11	11
14	—	10	10
15	—	9	9
16	—	8	8
17	—	—	23
18	—	—	22
19	—	—	21
20	—	—	20
21	—	—	19
22	—	—	18
23	—	—	17
24	—	—	16
25	—	—	31
26	—	—	30
27	—	—	29
28	—	—	28
29	—	—	27
30	—	—	26
31	—	—	25
(最低位) 32	—	—	24

フェーズでは、そのフェーズを実行している SCSI デバイスでのデータ・バスの駆動方法についての規定と、被選択 SCSI デバイス側でのデータ・バスのパリティ・チェックの方法に関する規定が追加になります。これらの追加規定は、異なったデータ・バス幅の SCSI デバイスを混在接続するために必要な条件です。

セレクション・フェーズのときイニシエータは、また、リセレクション・フェーズのときターゲットは、つぎのいずれかの方法でデータ・バスを駆動します。

- (1) データ・バス上で True にすべき SCSI ID ビットが含まれるバイトと、そのバイト位置よりも下位のすべてのバイトの全ビットを駆動します。このとき、駆動対象のバイトのパリティ・ビットは正しい値 (奇数パリティ) となるように駆動します。
- (2) SCSI ID ビットが含まれているバイトの位置に関わらず、自身がサポートしているデータ・バスのすべてのバイトを駆動します。このとき、全バイトのパリティ・ビットは正しい値 (奇数パリティ) となるように駆動します。

これらのどちらの方法を用いても、異なるデータ・バス幅の SCSI デバイスを混在接続しているとき、セ

レクシオン・フェーズまたはリセレクシオン・フェーズでは、データ・バスでパリティ・エラーが生じる場合があります。たとえば、A ケーブルの SCSI デバイスが P ケーブルの SCSI デバイスを選択するとき、前者はデータ・バスの DB7~0, P だけを駆動しますので、上位バイト: DB15~8, P1 は不当なパリティ・ビットの値をもった状態となります。

そこで、被選択側の SCSI デバイスは、つぎのルールでデータ・バスのパリティ・チェックを実行します。

- (1) DB7~0, P に対しては、つねにパリティ・チェックを実行します。
- (2) DB15~8, P1 は、DB31~8, P1, P2, P3 の範囲で True に駆動されているビットが少なくとも 1 ビットは存在するときだけ、パリティ・チェックを実行します。
- (3) DB23~16, P2 は、DB31~16, P2, P3 の範囲で True に駆動されているビットが少なくとも 1 ビットは存在するときだけ、パリティ・チェックを実行します。
- (4) DB31~24, P3 は、DB31~24, P3 の範囲で True に駆動されているビットが少なくとも 1 ビットは存在するときだけ、パリティ・チェックを実行します。

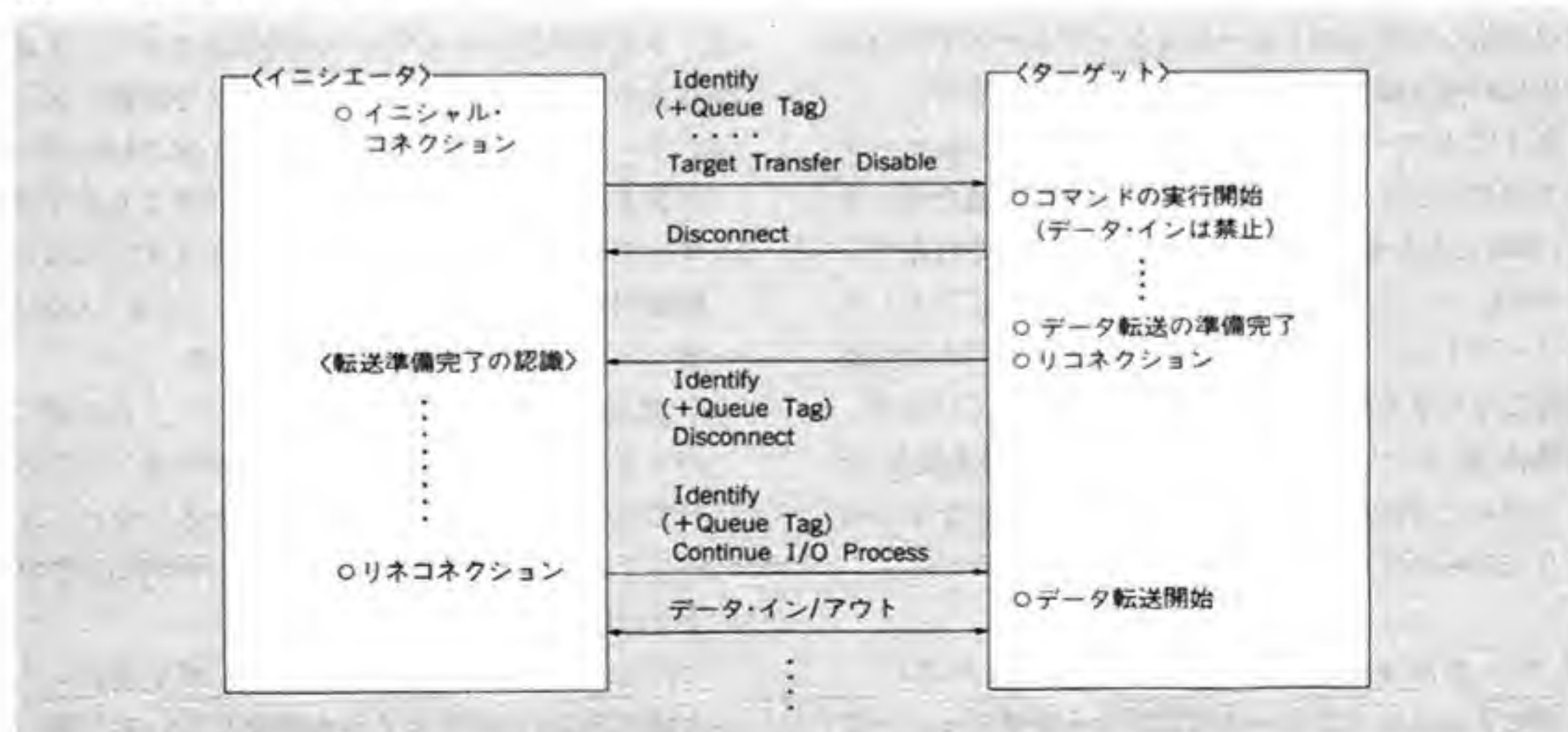
SCSI-3 インターロックド・プロトコル(SIP)では、SCSI-2 でのメッセージ・システムの規定に対して、いくつかのメッセージの追加、削除、変更を含む改良が行われることになっています。まだ詳細な仕様が確定していない部分もありますが、ここでは、すでに一部の製品での実用化が始まっている機能について解説します。

■ データ転送のタイミング制御

SCSI-1 や SCSI-2 の仕様では、イニシャル・コネクションで I_T_x_y ネクサスが確立された後の I/O プロセスの実行は、リセットやアボートなどの特殊な場合を除いて、すべてターゲットによって制御されます。したがって、イニシエータとターゲット間でデータ転送の実行をとまなう I/O プロセスの場合、いったんターゲットがディスコネクト処理を実行すると、その後のリコネクション処理とそれに続くデータ転送が開始されるタイミングは、ターゲットとロジカル・ユニットの動作に完全に依存することになります。

しかし、たとえば複数台のハード・ディスク装置を並列に接続して同時にデータ転送を行うことで、高速

図2 データ転送のタイミング制御



なデータ処理を可能にするディスク・アレイ・システムのようなアプリケーションが実用化されるにしたがって、データ転送の開始タイミングをイニシエータ側からも制御したいという要求がでてきました。

このような要求に応えるために、Continue I/O Process と Target Transfer Disable という二つのメッセージが用意されました。図 2 に、これらのメッセージを使用した典型的な例を示します。

■ Continue I/O Process

●メッセージ形式：1 バイト長(12h)

●転送方向：イニシエータ→ターゲット

●機能：このメッセージは、Target Transfer Disable メッセージと組み合わせて、イニシエータがデータ転送のタイミングをコントロールするために使用します。

I/O プロセスの起動時に、Target Transfer Disable メッセージによって、データ転送を実行するためのターゲットからのリコネクションを抑制している場合、イニシエータはそのネクサスをリコネクトして Continue I/O Process メッセージを発行することで、ターゲットに対してデータ転送を開始してもよいことを通知します。

▶メッセージの発行タイミング

イニシエータはリコネクションの実行時に、Identify メッセージと同一のメッセージ・アウト・フェーズで、このメッセージをターゲットに送信します。つまり、セレクション・フェーズの直後に実行されるメッセージ・アウト・フェーズでは、Identify メッセージ、キュー・タグ・メッセージ(タグ付き I/O プロセスの場合)、および Continue I/O Process メッセージが含まれることになります。

▶ターゲットの動作

このメッセージを受け取ったターゲットが、リコネクトされたネクサスについて、ただちに I/O プロセスを実行できる場合は、ターゲットはその処理、たとえばデータ・フェーズを実行します。

すぐに I/O プロセスの実行を開始できないときは、ターゲットは Disconnect メッセージを送信して、バスの開放を試みることができます。このときイニシエータは、Disconnect メッセージをリジェクトすることで、ターゲットに対して SCSI バスを結合したままで I/O プロセスを続行するように指示できます。

▶例外条件

イニシャル・コネクションのときにイニシエータがこのメッセージを送信した場合、ターゲットは、プロトコル上の重度なエラーとみなし、バス・フリー・フェーズに移行します。

▶Inquiry データ

このメッセージと Target Transfer Disable メッセージをサポートしているターゲットは、スタンダード Inquiry データのバイト 7・ビット 2(TranDis)ビットに“1”を表示します。

■ Target Transfer Disable

●メッセージ形式：1 バイト長(13h)

●転送方向：イニシエータ→ターゲット

●機能：このメッセージは、Continue I/O Process メッセージと組み合わせて、イニシエータがデータ転送のタイミングをコントロールするために使用します。

イニシエータは I/O プロセスの起動時にこのメッセージをターゲットに送信することで、その I/O プロセスの実行時に発生するデータ転送のためのリコネクション処理は、イニシエータの指示(Continue I/O Process メッセージ)によって行うことを示します。

▶メッセージの発行タイミング

イニシエータはイニシャル・コネクションのときに実行する最初のメッセージ・アウト・フェーズの最後のメッセージとして、このメッセージをターゲットに送信します。つまり、セレクション・フェーズの直後に実行されるメッセージ・アウト・フェーズで、Identify メッセージ、キュー・タグ・メッセージ(タグ付き I/O プロセスの場合)などを送信した後、そのメッセージ・アウト・フェーズの最後のメッセージとして、Target Transfer Disable メッセージを送信することになります。

▶ターゲットの動作

このメッセージを受け取ったターゲットは、通常の場合と同様にその後のバス・フェーズを実行して、イニシエータからコマンドを受け取ります。そして、つぎのように I/O プロセスを実行します。

(1) データ・アウト・フェーズをとまなう I/O プロセスのときは、通常の場合と同様に、ディスクコネクト処理が必要になるか、その I/O プロセスが完了するまでイニシャル・コネクションでデータ転送を実行することができます。

(2) データ・イン・フェーズをともなう I/O プロセスのときは、イニシャル・コネクションでデータ転送を実行することはできません。したがってターゲットはディスクコネクト処理を実行します。

(3) データ転送をともなわない I/O プロセスは、通常の場合と同様に、ディスクコネクト処理を行うか、またはその I/O プロセスを完了させます。

ここで、(1)または(2)でディスクコネクト処理を実行したとき、ターゲットはその I/O プロセスについてのその後のリコネクションを以下のように実行します。

(1) データ転送以外の処理(ステータス・フェーズなど)を実行するときは、通常の場合と同様にリコネクション処理を行い、その後続けて必要な処理を実行します。

(2) データ転送(データ・イン/アウト)が必要になったときは、ターゲットはつぎのシーケンスを実行して、イニシエータに対してデータ転送が可能になったことを通知します。

1. リセレクション・フェーズ
2. メッセージ・イン・フェーズ: Identify メッセージ(およびタグ付き I/O プロセスの場合は、Simple Queue Tag メッセージ)
3. メッセージ・イン・フェーズ: Disconnect メッセージ

(3) イニシエータが Disconnect メッセージをリジェクトしなかったとき、ターゲットはバス・フリー・フェーズに移行し、イニシエータからのリコネクションを待ちます。そして、Continue I/O Process メッセージを受け取ったときに、データ転送を実行します。

(4) イニシエータが Disconnect メッセージをリジェクトしたときは、ターゲットはこのリコネクションでデータ転送を実行することができます。

▶ Inquiry データ

このメッセージと Continue I/O Process メッセージをサポートしているターゲットは、スタンダード Inquiry データのバイト 7・ビット 2(TranDis)ビットに“1”を表示します。

参考・引用文献

- 1) *ASC, *Working Draft X3T9.2/855D Revision 12b*, "Information technology—SCSI-3 Parallel Interface", American National Standard—Accredited Standards Committee X3, June 7, 1993.
- 2) ASC, *Working Draft X3T9.2/856D Revision 3*, "Information technology—SCSI-3 Interlocked Protocol", American National Standard—Accredited Standards Committee X3, June 10, 1993.

すがや・せいichi 富士通㈱ファイル・システム第一技術

緊急企画

プラグ&プレイで急浮上の新規格

PCMCIA(PCカード)詳細解説

カード・サービス/ソケット・サービス/CIS/タブル/イネーブラ

岡村 周善

PCMCIA, 別名 PC カードの規格は、カードおよびスロットを規定する PC Card Standard と、デバイス・ドライバなどとのインターフェースを規定するソケット・サービス/カード・サービスに大別できる。日本の JEIDA も準じた規格であり、最近では PCMCIA/JEIDA スロット装備と明記する製品も増えている。PC カードの重要な概念の一つにそのカードがどんな属性なのかを示す CIS とタブルがある。また、PC カードをパソコン側からみたときには、CIS を読み出し、それに応じた設定にするイネーブラも重要である。ここでは、それらについて分かりやすく説明したあと、現状と問題点にふれる。(編集部)

ノート・パソコンやサブノート・タイプのハンディ・パソコンがつぎつぎと発売され、最近のものには必ず PCMCIA スロットが装備されています。また、それらの PCMCIA スロット用の IC カード・タイプの LAN カードや FAX/モデム・カード、さらにはハード・ディスク・カードまで発売されつつあります。しかし、販売現場やユーザ・サイドでの「PCMCIA 対応の IC カードがすべて、PCMCIA スロットをもつどのパソコンでもつかえるのか？」という疑問に対して、結局、メーカから個別の動作実績を聞き出して答えているのが実情です。そこで本稿ではノート・パソコンのカタログに記載されている PCMCIA スロットおよびカード・サービスやソケット・サービスと呼ばれるソフトウェアは何であるのかということについて解説いたします。

■ PCMCIA とは

まず最初に PCMCIA ということからきちんと定義

しておきましょう。PCMCIA とは「Personal Computer Memory Card International Association」の頭文字を集めた略称です。事務局は米国サニーベイルにあり、電気的な規格、物理的な規格、ソフトウェア面の規格などを個別に検討する分科会がいくつかあります。その分科会で討議された規格案や仕様案が会員の投票により承認されたものが一般にいわれている PCMCIA 規格とか PCMCIA 仕様ということになるわけです。PCMCIA は会費を払えば誰でも会員になれる任意団体で、そこで採択された規格や仕様は「会員はこの規格に沿った製品を出しましょう」というガイドラインでしかありませんので、互換性のうえで少々問題が生じています。

PCMCIA の規格は厚さ 5 cm くらいの規格書にまとめられており図 1 のような構成になっています。なお PCMCIA では PCMCIA 規格に準じた IC カードのことを「PC カード」と呼んでいますので、本稿でも以降は PC カードと呼ぶようにします。規格は大きくわけてカードおよびスロットを規定する「PC Card Standard」と上位のソフトウェア(デバイス・ドライバやユーティリティ・プログラムなど)とのソフトウェア的なインターフェースを規定する「ソケット・サービス仕様」、「カード・サービス仕様」とに分かれています。

このほか、PC カード・タイプのハード・ディスクを従来の IDE ドライブ用のインターフェースに合わせるためのプロトコル、I/O レジスタなどを規定した「PC Card ATA 規格」、マルチメディア用のイメージ・データなどを記録する大容量のフラッシュ・メモリ・カードに関する「PCMCIA AIM 規格」の 2 種類が追加されています。これらの ATA 規格と AIM 規

格については別の機会にゆずります。

なおパソコンのカタログなどでは「PCMCIA 2.0/ JEIDA 4.1」というように並記されていますが、これは日本国内にも PCMCIA のような役割をしている組織があり、その規格にも準じていることをしめています。日本国内では(社)日本電子工業振興協会(Japan Electronics Industry Development Association)内に IC カードに関する委員会があり PCMCIA と同様な規格を定めています。PCMCIA と JEIDA は互いに規格案を交換しあって統一するように協力しています。それぞれ互いのバージョンに互換性があり、その関係は下記のようになっています。

PCMCIA 1.0 ≒ JEIDA 4.0
(物理的、電氣的仕様ののみ)

PCMCIA 2.0 ≒ JEIDA 4.1
(物理的、電氣的仕様ののみ)

PCMCIA 2.1 ≒ JEIDA 4.2^{a)}

注) JEIDA 4.2 でソケット・サービス、カード・サービスが追加されました。

図1 PCMCIA の規格

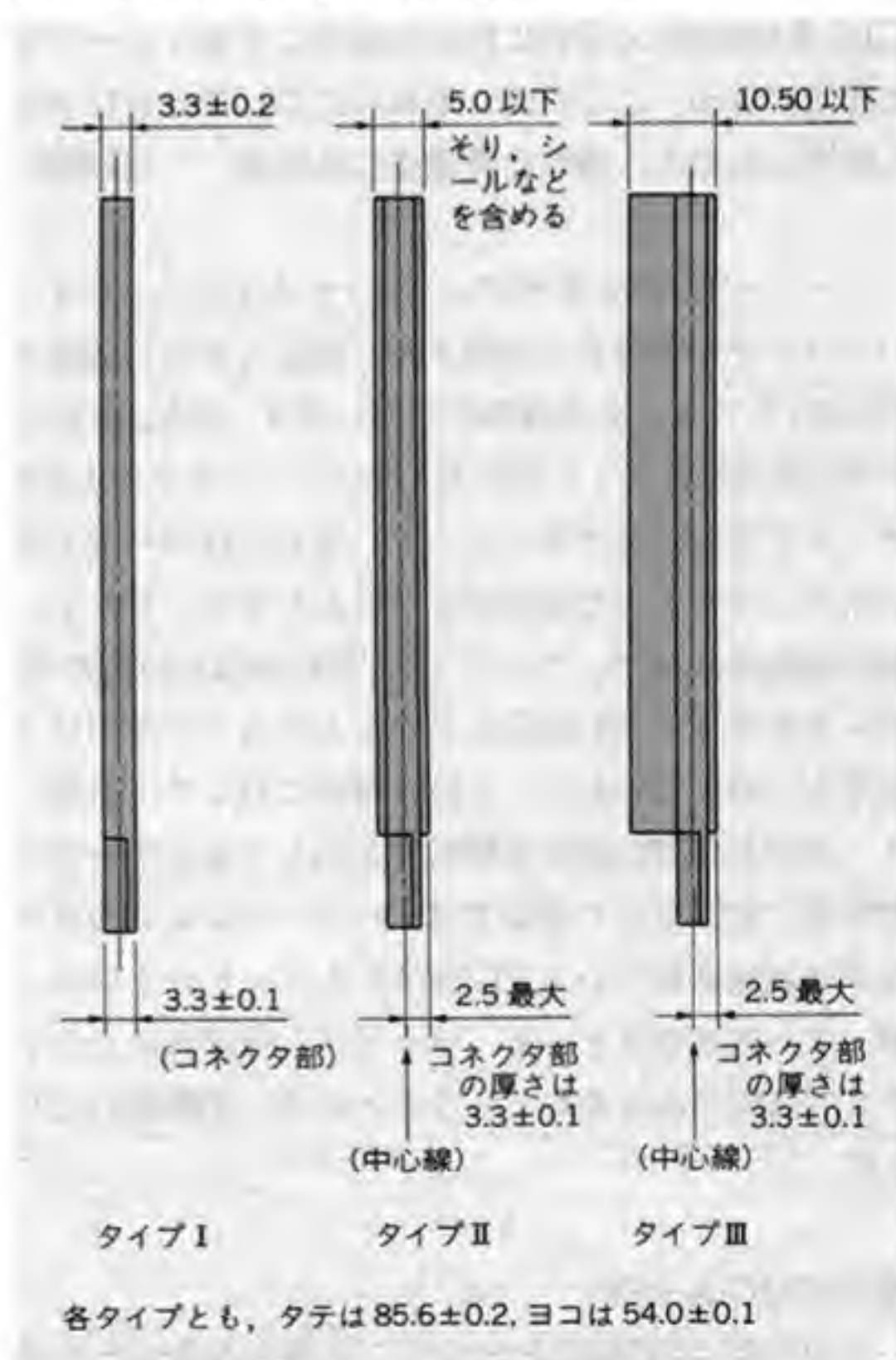


1 PCMCIA スロット

■ スロットのサイズについて

PCMCIA の PC カードの規格は、まずカードの物理仕様(サイズなど)から記述されています。PC カードのサイズは厚さ(縦横の大きさは同じ)により図2のように3種類(最近では大容量ハード・ディスクのためにタイプIVという 15 mm 厚のものも提案されているので、4種類)存在します。このうちタイプ I はメモリ・カードで使用されている 3 mm 厚のサイズです。タイプ II はそれよりも少し中央部が厚く 5 mm 厚となっています。タイプ II は LAN カードやモデム・カードなどの I/O カードでよく使用されています。タイプ III はタイプ II 用の 2 連スロットにおさまるよう(図3)に 10.5 mm 厚に決められており、ハード・ディスクで使用されています。ただし、PCMCIA 2.1 ではタイプ

図2 PCMCIA の各タイプの厚さ(単位: mm)



I, IIは規格化されていますが、タイプIIIは推奨仕様です。

これら以外にも PC カードの形状にはいろいろと細かな工夫がなされています。カードを裏返しにスロットに挿入しないようにエッジ部の左右の切り込みの形が異なります。さらに 3.3 V で動作するカードを間違えて 5 V が供給されているスロットに挿入して壊さないよう、この切り込みの形状で区別しようという案が現在 PCMCIA で協議されています。

■ PC カード・スロットの信号

PCMCIA の規格書では物理仕様のつぎにスロットの信号について説明しています。PC カードとの接続に使用されているコネクタは 34×2 列の 68 ピンのコネクタです。このコネクタの物理仕様その他は本稿の趣旨からはずれるので割愛し、コネクタの各端子の信号について説明します。

コネクタの信号配列は表 1 のようにメモリ・カード用と I/O カード用に 2 種類規定されています(最近では ATA ハード・ディスク用にもう 1 種類規定されている)。メモリ・カード用の信号配列は PCMCIA 1.0/ JEIDA 4.0 の時代に規格化されたもので、I/O カードに必要な信号線(IORD, IOWR, IRQ など)がないためメモリ・カードしか使用できません。したがって、パソコンのカタログなどを調べる場合にはこの PCMCIA/ JEIDA のバージョン番号に注意が必要です。「PCMCIA 1.0/ JEIDA 4.0」の場合にはメモリ・カード専用スロットということになり、I/O カードは使用できません。PCMCIA 2.0/ JEIDA 4.1 になって I/O カード用の信号配列が規格化され、メモリ・カード以外にも I/O カードが使用できるようになりました。表 1 の信号配列をみくらべると 6 個の端子の信号が異

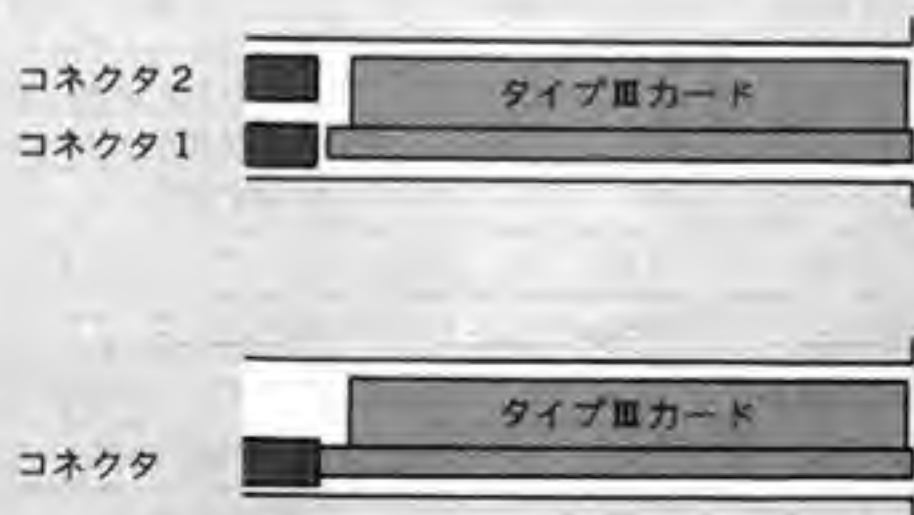
なっています。たとえば 16 番の端子がメモリ・カード用では「RDY/−BSY」信号に割り当てられていますが、I/O カード用では「−IREQ」(割り込み要求信号)に割り当てられています。

PCMCIA 2.x(2.0 以降)でもっとも重要なことは PCMCIA 1.0 との完全なアップ・コンパチビリティを実現するため、このメモリ・カード用の信号と I/O カード用の信号の両方を同じスロットで供給しなければいけないということです。したがって、同じ端子番号で信号が異なるものについてはスロット側 PC カード側の双方で切り替えることが必要になります。この切り替えはスロット内に挿入されているカードの種別(メモリ・カードか I/O カードか)を判断してソフトウェアで実行しなければなりません。つまり、最初スロット側の信号はメモリ・カード用に設定されており、I/O カード側もそれに合わせて最初はメモリ・カードと同じ信号配列に自分自身を設定しています。I/O カード挿入検出後イネーブラやドライバ・ソフトウェアがスロット側、I/O カード側双方の信号設定を I/O カード用に設定しなおします(カード・コンフィギュレーション)。

PCMCIA スロットの信号はデスクトップ・パソコンの ISA バスの信号とほぼ同じような信号が配置されています。ISA バスにはない信号を表 2 にまとめてみました。CE₁, CE₂はカードに対するイネーブル信号、CD₁, CD₂はカードの検出信号です。メモリ・カードのための WP(ライト・プロテクト)信号や EEPROM のための PGM 信号も用意されています。

EEPROM やフラッシュ・メモリ用の V_{pp}(書き込み電源)出力はデフォルトでは 5 V で、ソフトウェアにより 12 V に変更できるように規定されています。メモリ・カード用にはほかにカード内のバッテリー電圧の

図 3 タイプIIIカードの実装



2連スロットの場合、片方が使用できなくなる。また下側のスロットの論理ソケット No. が 1 の場合 (FM-V など) と上側のスロットの論理ソケット No. が 1 の場合 (ThinkPad, Pronote jet など) の場合がある。

コネクタが一つだけで空間のみタイプIII用になっているパソコンの場合(東芝 DynaBook V486A など)

表1 PCMCIA PC カード, スロット・コネクタ信号配列

<表の見方>

1) Mem はメモリ・カード・インターフェース・モード(電源 ON 後の初期値)

2) I/O は I/O カード・インターフェース・モード(コンフィギュレーション実行後)

3) 方向は PC カードへの入力方向を I, PC カードからの出力方向を O で示す。双方向の場合は I/O で示す。

端子番号	Mem	I/O	方向	機 能	端子番号	Mem	I/O	方向	機 能
1	GND	GND	—	グラウンド	39	D ₁₃	D ₁₃	I/O	データ・ビット 13
2	D ₁	D ₅	I/O	データ・ビット 3	40	D ₁₄	D ₁₄	I/O	データ・ビット 14
3	D ₄	D ₄	I/O	データ・ビット 4	41	D ₁₅	D ₁₅	I/O	データ・ビット 15
4	D ₅	D ₅	I/O	データ・ビット 5	42	CE ₂	CE ₂	I	カード・イネーブル
5	D ₆	D ₆	I/O	データ・ビット 6	43	RFSH	RFSH	I	リフレッシュ
6	D ₇	D ₇	I/O	データ・ビット 7	44	RFU	IORD	I	リザーブ:メモリ IORD: I/O
7	CE ₁	CE ₁	I	カード・イネーブル	45	RFU	IOWR	I	リザーブ:メモリ IOWR: I/O
8	A ₁₀	A ₁₀	I	アドレス・ビット 10	46	A ₁₇	A ₁₇	I	アドレス・ビット 17
9	OE	OE	I	出力イネーブル	47	A ₁₈	A ₁₈	I	アドレス・ビット 18
10	A ₁₁	A ₁₁	I	アドレス・ビット 11	48	A ₁₉	A ₁₉	I	アドレス・ビット 19
11	A ₉	A ₉	I	アドレス・ビット 9	49	A ₂₀	A ₂₀	I	アドレス・ビット 20
12	A ₈	A ₈	I	アドレス・ビット 8	50	A ₂₁	A ₂₁	I	アドレス・ビット 21
13	A ₁₃	A ₁₃	I	アドレス・ビット 13	51	V _{cc}	V _{cc}	—	カード用電源
14	A ₁₄	A ₁₄	I	アドレス・ビット 14	52	V _{pp1}	V _{pp1}	—	フラッシュ・メモリ書き込み用電源 I/O モードでは周辺回路用にも使用可
15	WE/PGM	WE/PGM	I	ライト・イネーブル	53	A ₂₂	A ₂₂	I	アドレス・ビット 22
16	RDY/BSY	IREQ	O	Ready/Busy :メモリ・モード 割り込み要求 :I/O モード	54	A ₂₃	A ₂₃	I	アドレス・ビット 23
17	V _{cc}	V _{cc}	—	カード用電源	55	A ₂₄	A ₂₄	I	アドレス・ビット 24
18	V _{pp1}	V _{pp1}	—	フラッシュ・メモリ書き込み用電源 I/O モードでは周辺回路用にも使用可	56	A ₂₅	A ₂₅	I	アドレス・ビット 25
19	A ₁₆	A ₁₆	I	アドレス・ビット 16	57	RFU	RFU		リザーブ
20	A ₁₅	A ₁₅	I	アドレス・ビット 15	58	RESET	RESET	I	カード・リセット
21	A ₁₂	A ₁₂	I	アドレス・ビット 12	59	WAIT	WAIT	O	Wait 要求信号
22	A ₇	A ₇	I	アドレス・ビット 7	60	RFU	INPACK	O	リザーブ:メモリ IORD 応答信号: I/O データ・バス・バッファ 制御用に使用
23	A ₆	A ₆	I	アドレス・ビット 6	61	REG	REG	I	アトリビュート・メモリ・ セレクト信号 I/O モードでは I/O 空間 セレクト信号としても 使用される
24	A ₅	A ₅	I	アドレス・ビット 5	62	BVD ₂	SPKR	O	カード内蔵バッテリー電圧 検出信号:メモリ カード・モデムなどのスピー カ駆動信号: I/O
25	A ₄	A ₄	I	アドレス・ビット 4	63	BVD ₁	STSCHG	O	カード内蔵バッテリー電圧 検出信号:メモリ BVD, Rdy/BSY, WP の 信号に変化があったこと を示す信号
26	A ₃	A ₃	I	アドレス・ビット 3	64	D ₈	D ₈	I/O	データ・ビット 8
27	A ₂	A ₂	I	アドレス・ビット 2	65	D ₉	D ₉	I/O	データ・ビット 9
28	A ₁	A ₁	I	アドレス・ビット 1	66	D ₁₀	D ₁₀	I/O	データ・ビット 10
29	A ₀	A ₀	I	アドレス・ビット 0	67	CD ₂	CD ₂	O	カード検出信号
30	D ₀	D ₀	I/O	データ・ビット 0	68	GND	GND	—	グラウンド
31	D ₁	D ₁	I/O	データ・ビット 1					
32	D ₂	D ₂	I/O	データ・ビット 2					
33	WP	IOIS16	O	Write Protect:メモリ 16 ビット I/O ポート応答 信号: I/O					
34	GND	GND	—	グラウンド					
35	GND	GND	—	グラウンド					
36	CD ₁	CD ₁	O	カード検出信号					
37	D ₁₁	D ₁₁	I/O	データ・ビット 11					
38	D ₁₂	D ₁₂	I/O	データ・ビット 12					

表2 PCMCIA の信号で ISA がない信号

〈表の見方〉

モード欄の「メモリ」はメモリ・インターフェース・モード時を示し、「I/O」はI/O インターフェース時を示す。「共通」は両方のモードに共通であることを示す。

信号名	モード	機 能
CD ₁ , CD ₂	共通	カード検出信号。カード内部でGNDに接続されている。スロット側でプルアップしているの、カードが挿入されると“L”になり、抜くと“H”になる。
BVD ₁ , BVD ₂	メモリ	カード内のバックアップ用バッテリーの電源電圧の状態を出力する。
V _{pp1} , V _{pp2}	共通	フラッシュ・メモリのための書き込み電源を供給するために使用される。I/Oカードの場合には、このほかV _{cc} 以外の電圧の電源として使用可。しかしいずれの場合でも、CISを読み出して電圧を確認するまではV _{cc} と同じ電圧が供給される。
CE ₁ , CE ₂	共通	カードに対するセレクト信号(“L”アクティブ)。CE ₁ は偶数番地、CE ₂ は奇数番地に対するセレクト信号。
OE	共通	カード内のメモリに対する読み出しストロブ信号。アトリビュート・メモリやCCRに対するリード時もアクティブにされるが、I/Oアクセス時にはアクティブにならない。ISAバスの-MEMRに相当する。
WE	共通	カード内のメモリに対するライト・ストロブ信号。アトリビュート・メモリやCCRに対するライト時もアクティブにされるが、I/Oアクセス時にはアクティブにならない。ISAバスの-MEMWに相当する。
RDY/BSY	メモリ	カード内のメモリに対するリードあるいはライト処理実行中の場合(BSY)でデータ転送要求に対して応答できない場合に“L”にアサートされる。
IREQ	I/O	カードからの割り込み要求を出力する。ISAバスとは異なり、負論理で出力される。
WP	メモリ	メモリ・カード後部のライト・プロテクト・スイッチの状態を出力する。
IOIS16	I/O	ISAバスのIOCS16に相当する。
WAIT	共通	ISAバスのIOCHRDYに相当する。
REG	共通	アトリビュート・メモリに対するセレクト信号。本信号のアサート時、アトリビュート・メモリに対してはOEおよびWEでアクセスし、I/Oに対してはIORD、IOWRでアクセスする。
INPACK	I/O	CEおよびIORDの両方がアサートされ、かつアドレスがカード内のI/Oポートと一致するときに出力される。データ・バス・バッファのコントロール用に使用する。
SPKR	I/O	カード上のドライバでホスト(パソコン)上のスピーカを駆動するための信号出力。
STSCHG	I/O	カード内のBVD、Rdy/BSY、WPのいずれかの信号が変化したことを通知するための出力信号。

状態を検出するための信号が用意されています。I/Oカード用ではIORD、IOWR、IREQ信号などのほかにカード側からのバッテリー電圧低下、ライト・プロテクト、BUSYなどの信号が変化したことを示すSTSCCHG信号、モデム・カードなどのためのスピーカ駆動用のSPKR信号が用意されています。

I/Oカードからの割り込み要求は-IREQ信号を通してスロットを通りPCMCIAコントローラLSIに伝えられます。コントローラLSIはこのIRQ信号をパソコン・マザーボード上の割り込みコントローラにルーティングして伝えます(図4参照)。PCカードからのIRQをどのIRQに割り当てるかというルーティング情報はあらかじめソフトウェアでPCMCIAコントローラLSI内部のレジスタに設定しておきます。

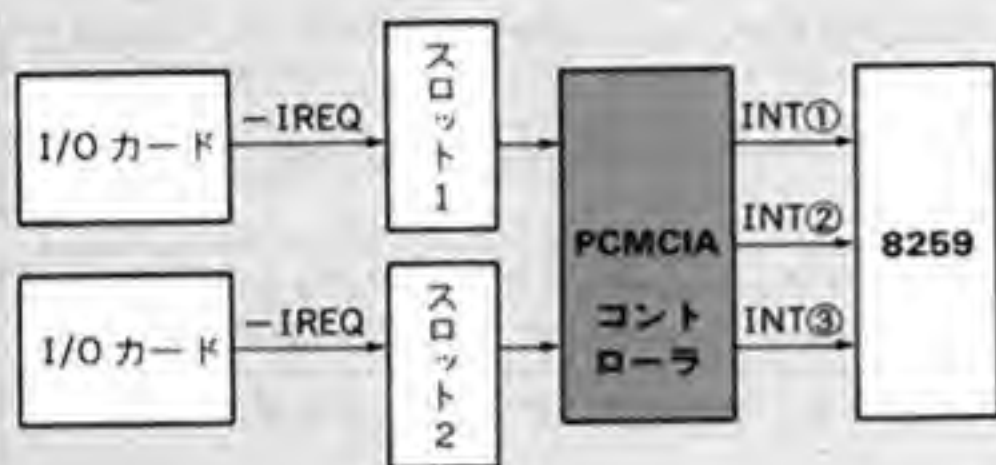
PCMCIAスロットのデータ・バス信号は16ビットもしくは8ビットですが、アドレス・バス信号はA₀か

らA₂₅までの26ビットあるので64Mバイトのアドレス・スペースがあります。実際にはこんなにたくさんアドレス信号を使っているわけではなく、とくにI/OカードではPC/AT(ISAバス)のI/Oアーキテクチャに合わせるためアドレスの下位10ビットのみデコードしている製品がほとんどです。そのためPCMCIAではI/Oアドレス・スペースのデコード方法について図5のように2種類のモードを規定しています。インディペンデント・モードではカード側でCEがアサートされればすべて応答するのに対し、オーバラッピング・モードではアドレス入力の下位の一部(通常はISAバスと同じ10ビット)をデコードします。一般的にはこのオーバラッピング・モードのほうが使用されています。メモリ・カードのほうもDOSベースでは何らかのウィンドウを設定しないと64Mバイトというような広大な空間を使用することはできません。

しかし、メモリ・ウィンドウに関してはPCカード側ではとくに規格化されておらず、カード・サービスなどで解決すべく提案されています。

PCMCIAの規格では、当然スロットの信号タイミングについても規定されていますが、本稿では割愛します。

図4 IRQのルーティング



①、②、③は2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15のいずれか。

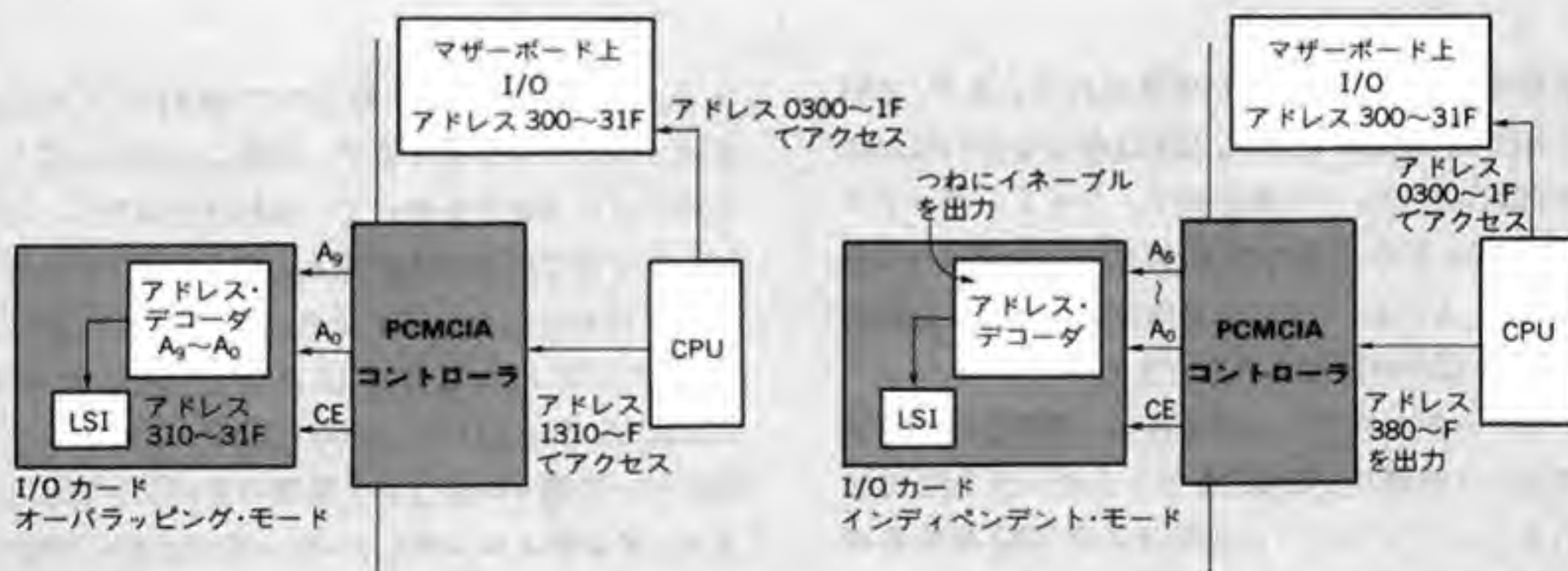
INT①はスロット1からの-IREQを伝える。INT②はスロット2からの-IREQを伝える。INT③はカードの挿入、抜去などのイベントを通知するためにPCMCIコントローラ自身が発行する。①、②、③に相当するINT番号はPCMCI内のレジスタに設定する。

■ カード・コンフィギュレーション

スロット側のメモリ・インターフェース・モードとI/Oインターフェース・モードの切り替えはPCMCIコントローラLSI内のレジスタを操作することによって行います。PCMCIの規格ではこのPCMCIコントローラLSIに関しては何も規定していませんが、カード側のインターフェース・モード切り替え方法については当然ながら規定されています。カード側のインターフェース・モード設定のことをPCMCI 2.xでは「Card Configuration」と称しています。カード・コンフィギュレーション用にカード内に図6, 7, 8, 9のような特別なレジスタを用意しておくことが規定されています。

そして、当然これらのレジスタはインターフェース・モードに関係なく(メモリ・モードでもI/Oモードでも)リード/ライトできることが規定されています。これらのレジスタ類はまとめてCCR(Card Configuration Register)と呼ばれています。このレジスタの存在するアドレス、何を書き込めばよいかという情報、また4個のレジスタのうち、どれとどれをもっているかという情報はすべてつぎに説明するCISのなかに記述されています。

図5 オーバーラッピング・モードとインディペンデント・モード



オーバーラッピング・モードを使用すれば、 $A_9 \sim A_0$ しかデコードしていないカードを、上記の例のように $A_{15} \sim A_{10}$ をデコードしてマッピングすることができる。したがって他のI/Oと $A_9 \sim A_0$ が重なっていても使用できる。

インディペンデント・モードでは、カードはCEのみでセレクトされるため、I/Oカードがもっているアドレス・デコードのサブセットを無視してホストCPU側とPCMCIコントローラ間だけでアドレスを決めることができる。ただし、ベース・アドレスの A_6 から A_0 はすべて0であること。

カード・コンフィギュレーションのフローを図10に示します。パソコン上のソフトウェアはスロットに挿入されたPCカードのアトリビュート・メモリを読み出し、メモリ・カードであるのか、I/Oカードであるのかチェックします。I/Oカードであった場合はスロット側の信号をI/Oインターフェース・モードに切り替えた後、PCカードのアトリビュート・メモリの特定アドレスにあるCCORやPRR(Pin Replacement Register)に決められた値(カードにより異なる)を書き込むことによりカード側のインターフェースをI/Oインターフェース・モードに切り替えます。これで初めてI/Oカードとして使えるようになるわけです。

2 PCカードのメタフォーマット(CIS)について

■ CIS とは

さて前章でも説明しましたが、PCMCIA 2.xのキーポイントはPCカード側、スロット側のどちらも最初は(デフォルト状態)PCMCIA 1.0のメモリ・カード・インターフェースであるということです。したがってI/Oカードを使用するためにはソフトウェアで双方の

図6 CCOR(カード・コンフィギュレーション・オプション・レジスタ)

機能：カードのコンフィギュレーションを行うためのレジスタでI/Oカード(コンフィギュレーション可能なカード)には必ず存在しなければならない。

所在：アトリビュート・メモリの先頭からCCRのオフセット・アドレス+0。

	b7	b6	b5	b4	b3	b2	b1	b0
R/W	SRESET	LevelIRQ	コンフィギュレーション・インデックス					

b7：SRESET

本ビットに1をセットするとカードがリセット状態になる。その後、0に戻すとカードはパワーオン・リセットまたはハードウェア・リセット後の状態に復帰する。なお、本ビットはパワーオン・リセットまたはハードウェア・リセット後は0になっている。

b6：LevelIRQ

本ビットを1にするとレベル・モードIRQが選択され、0にセットするとパルス・モード(エッジ・モード)IRQがセットされる。リセット後の初期値はカードにより不定。

b5…b0：コンフィギュレーション・インデックス

CIS内よりソフトウェア(イネーブラやドライバ)で選択したコンフィギュレーション・エントリ・タプルのインデックス番号を書き込む。インデックス番号0はI/Oカード・モードをディスエイブルにし、メモリ・カード・モードに設定する。

図7 CCSR(カード・コンフィギュレーション・ステータス・レジスタ)

機能：カードのコンフィギュレーション状態を示すためのレジスタ。本レジスタは必須ではないが、カードがAudioなどを使用している場合には必要。

所在：アトリビュート・メモリの先頭からCCRのオフセット・アドレス+2。

	b7	b6	b5	b4	b3	b2	b1	b0
リード	Changed	SigChg	IOis8	Rsvd0	Audio	PwrDwn	Intr	Rsvd0
ライト	—	SigChg	IOis8	Rsvd0	Audio	PwrDwn	—	Rsvd0

b7、b1はカードによってセット/リセットされるので、ソフトウェアでは操作できない。

b7：Changed

本ビットはPRRの上位4ビットのうちのどれか一つ以上が1にセットされていることを示す。本ビットが1、SigChg(b6)も1にセットされていてカードがコンフィギュレーションされていればSTSCHG(カード・コネクタの63番端子)が“L”にアサートされる。

b6：SigChg

本ビットはSTSCHG信号をイネーブル、ディスエイブルにするために使用する。1をセットするとイネーブルになり、0をセットするとディスエイブルになる。

b5：IOis8

16ビット・レジスタをもつカードにD7からD0のみのアクセスでアクセスすることを指定する。8ビット・データ・バスのホストに16ビット・データ・バスをもつカードを接続する場合に使用する。

b4：Rsvd0

リザーブ・ビット。必ず0にすること。

b3：Audio

本ビットを1にセットするとSPKR(カード・コネクタの62番端子)がイネーブルになる。0をセットするとディスエイブルになる。

b2：PwrDwn

パワー・ダウン・モードをもつカードの場合、本ビットを1にセットするとカードがパワー・ダウン・モードに移行する。0に戻すと通常モードに復帰する。

b1：Intr

カードがIREQを出力している場合1にセットされる。ソフトウェアによって割り込み要因を取り除くと0にリセットされる。

b0：Rsvd0

リザーブ・ビット。必ず0にすること。

図8 PRR(ピン・リブレースメント・レジスタ)

機能：カード・コネクタの 16, 33, 62, 63 番端子に現れるカードの状態信号を取り出すためのレジスタ。Ready/Busy, WP, BVD などの情報が必要な場合は本レジスタをもっていなければならない。カードが I/O カード・モードにコンフィギュレーションされていて上記の各端子が IREQ, IOIS16, SPKR, STSCHG として使用されている場合、Ready/Busy, WP, BVD の各情報は本レジスタの下位 4 ビットを読み出すことによって知ることができる。

所在：アトリビュート・メモリの先頭から CCR のオフセット・アドレス+4。

	b7	b6	b5	b4	b3	b2	b1	b0
リード	CBVD1	CBVD2	CRdy/Bsy	CWP	RBVD1	RBVD2	RRdy/Bsy	RWP
ライト	CBVD1	CBVD2	CRdy/Bsy	CWP	Maskb7	Maskb6	Maskb5	Maskb4

b7: CBVD1

RBVD1 の状態が変化(0 から 1 または 1 から 0)すると 1 にセットされる。
Maskb7 に 1 が書き込まれていなければ本ビットに 1 または 0 を書き込むことができる。

b6: CBVD2

RBVD2 の状態が変化(0 から 1 または 1 から 0)すると 1 にセットされる。
Maskb6 に 1 が書き込まれていなければ本ビットに 1 または 0 を書き込むことができる。

b5: CRdy/Bsy

RRdy/Bsy の状態が変化(0 から 1 または 1 から 0)すると 1 にセットされる。
Maskb5 に 1 が書き込まれていなければ本ビットに 1 または 0 を書き込むことができる。

b4: CWP

RWP の状態が変化(0 から 1 または 1 から 0)すると 1 にセットされる。
Maskb4 に 1 が書き込まれていなければ本ビットに 1 または 0 を書き込むことができる。

b3: RBVD1

リード時：カード内部の BVD1 の状態を示す。
ライト時：1 を書き込むと CBVD1 ビットが書き込み禁止になる。0 を書き込むと解除される。

b2: RBVD2

リード時：カード内部の BVD2 の状態を示す。
ライト時：1 を書き込むと CBVD2 ビットが書き込み禁止になる。0 を書き込むと解除される。

b1: RRdy/Bsy

リード時：カード内部の Rdy/Bsy の状態を示す。
ライト時：1 を書き込むと CRdy/Bsy ビットが書き込み禁止になる。0 を書き込むと解除される。

b0: RWP

リード時：カード内部の WP スイッチの状態を示す。
ライト時：1 を書き込むと CWP ビットが書き込み禁止になる。0 を書き込むと解除される。

設定を変更する(コンフィギュレーション)作業が必要です。図10に示すように I/O カードも最初はメモリ・カードとしてふるまいます。スロット側もメモリ・カード・インターフェースに設定されたままなので、このままではどんな種類のカードがスロットに挿入されているかを判断することができません。そこで PCMCIA 2.x では PC カードの種類にかかわらず、そのカードがどんなカードであるか(属性)を記録した不揮発性メモリをカード内にもつことが定められています。このメモリの読み出しは、PC カードとスロットがメモリ・カード・インターフェース・モードにあらうが、I/O カード・インターフェース・モードにあらうが関係なく実行できます。PCMCIA 2.x ではこのメモリのことを「ATTRIBUTE memory」と呼んでいます。そしてこのアトリビュート・メモリ上に書き込まれている属性データ全体のことを CIS(Card Information Structure)、CIS 内の個々のデータ構造体のことを Tuple(JEIDA の用語ではタプル)と呼んでいます。もちろん CIS やタプルの定義については細かく規定されています。

■ タプル

前記のように CIS はタプルの集合で構成されており、アトリビュート・メモリの先頭(オフセット・アドレス 0)から順にタプルが詰め込まれています。個々のタプルのフォーマットは図11のように先頭に 1 バイトのタプル・コード、つぎに 1 バイトのオフセット値(つぎのタプル・コードの先頭へのオフセット値)があり、そのつぎからタプルのデータ部が始まります。一つのタプル全体のサイズは 256 バイト以下でなければなりません(つぎのタプルへのオフセット値フィールドが 1 バイトしかないため)。タプル・コードは PCMCIA 2.1 では表 3 (p. 176)のように 30 種類あり、四つのレイヤに区分されています。このうちコード 1Eh から 23h までの 6 個のタプルは 2.1 で追加されたものです。とくにコード 22h の機能拡張タプルでは FAX/モデム・カードに関する詳細な記述まで規定されています。たとえば、カードに内蔵している UART の種類、フロー制御の方法、モデムのサポート手順、コマンド体系、変調方式などの記述方法が規定されています。ただ各 PC カードはこれらのタプル・コードを

図9 SCR(ソケット・コピー・レジスタ)

機能：同じカードまたは類似のカードを同時に複数枚使用する際にカードの区別を行うためにソフトウェアで使用できるレジスタ。必須ではない。
 所在：アトリビュート・メモリの先頭から CCR のオフセット・アドレス+6.

	b7	b6	b5	b4	b3	b2	b1	b0
リード	Reserved	Copy Number			Socket Number			
ライト	Reserved	Copy Number			Socket Number			

b7：Reserved
 ライト時、つねに 0 を書き込む。
 b6, b5, b4：Copy Number
 コンフィギュレーションを実行した順番に 0 から始まる数を書き込む。
 b3, b2, b1, b0：Socket Number
 そのカードが現在入っているソケット(スロット)の番号を書き込む。
 物理的に 1 番目のソケットの番号は 0.

図12 モデム・カードの CIS 例

このような順序でダブルが書き込まれている。

CISTPL_DEVICE デバイス情報ダブル
CISTPL_VER_1 レベル 1 バージョン/製品情報ダブル
CISTPL_CONFIG コンフィギュレーション・ダブル
CISTPL_CFTABLE_ENTRY コンフィギュレーション・エントリ・ダブル (インデックス 20h)
CISTPL_CFTABLE_ENTRY コンフィギュレーション・エントリ・ダブル (インデックス 21h)
CISTPL_CFTABLE_ENTRY コンフィギュレーション・エントリ・ダブル (インデックス 22h)
CISTPL_CFTABLE_ENTRY コンフィギュレーション・エントリ・ダブル (インデックス 23h)
CISTPL_NO_LINK ノーリンク・ダブル
CISTPL_END ダブル・チェーン終了ダブル

すべて使用しなければならないということではないので、実際には必要なものを選択して使用すればよいことになっています。

■ CIS の構成例とダブルの詳細

実際の PC カードの CIS の例として図12 に一般的なモデム・カードの CIS を紹介します。このカードでは 6 種類のダブル・コードが使用されています。

CIS の先頭はデバイス情報ダブル(CISTPL

図10 カード・コンフィギュレーション・フロー

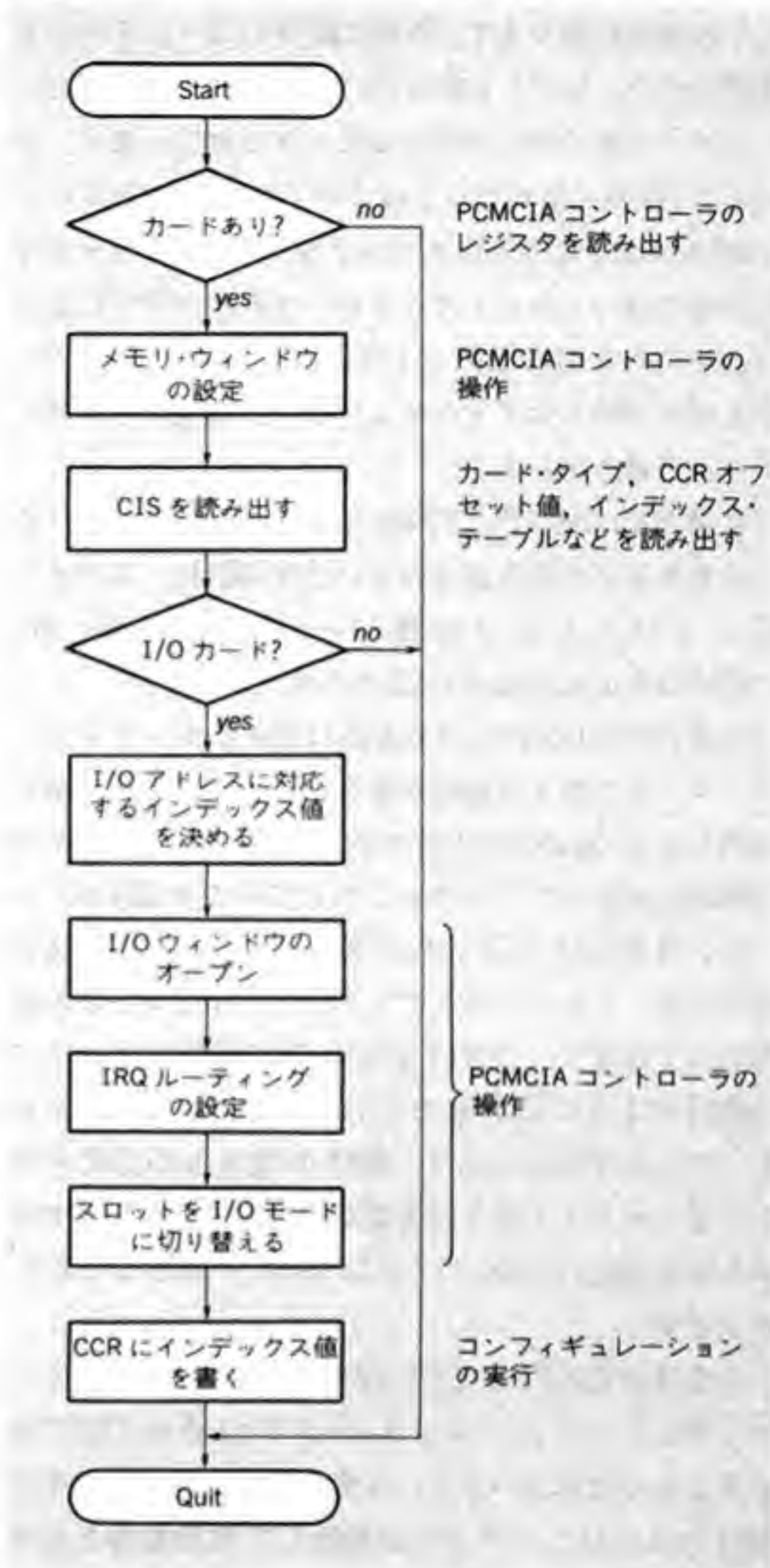
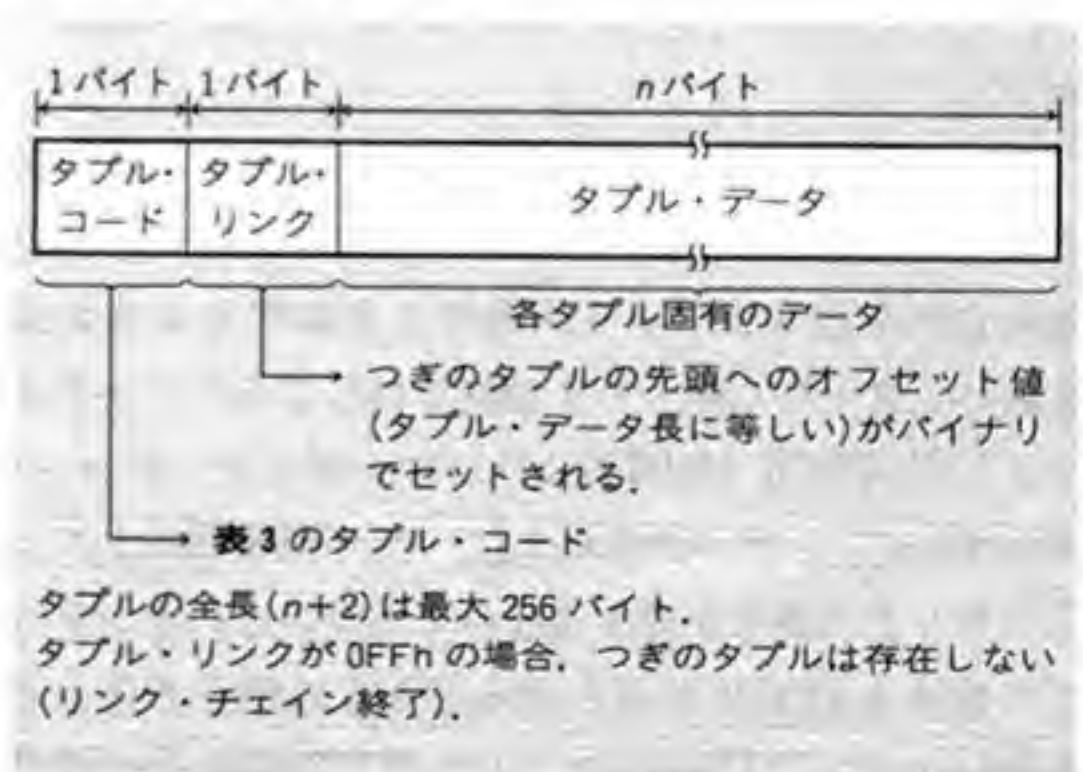


図11 ダブル・フォーマット



DEVICE)です。このタプルは必ずアトリビュート・メモリの先頭にあります。内容は図13のように4バイトで構成され、3バイト目にデバイスID、4バイト目にデバイス・サイズ・バイトがセットされています。デバイスIDの上位4ビットはデバイス・タイプ・コードで図13のようにきめられています。モデム・カードなどの場合は、メモリ・カードのいずれのタイプにも入りませんので0Dhがセットされています。デバイス・サイズ・バイトはアトリビュート・メモリのサイズとブロック数を示します。

2番目はCISTPL_VERS_1(15h)で製品のメーカー名や型名などが書き込まれています(図14)。3バイト目と4バイト目は仕様バージョンで04h, 01h(PCMCIA 2.xの場合)に決められています。

3番目はCISTPL_CONFIG(1Ah)でコンフィギュレーションに関する情報が書き込まれています。前に説明したCCRの所在(オフセット・アドレス)などの情報が記述されているのがこのタプルです(図15)。5バイト目からはCCRのベース・アドレスが書き込まれており、リトル・エンディアン・フォーマットが使用されています。つぎのTPCC_RMASKフィールドは図15のようにCCORをb₀にして何個のレジスタをもっているかを示します。図15の場合はCCORのアトリビュート・メモリの先頭からのオフセット・アドレスは03f0h, CCSRのそれは03f2hであることがわかります。

4番目はCISTPL_CFTABLE_ENTRY(1Bh コンフィギュレーション・エントリ・タプル)でCCORに何を書き込んだらよいかという情報が記述されています。図16のようにこのタプルは連続して複数個書き込まれています。モデム・カードの場合には通常、カードに内蔵しているUARTをPC-BIOSのCOM1, 2, 3, 4のいずれかに割り付けるので、このタプルは4個あります。

これらのうちの先頭のタプルは他のタプルに比べて書き込まれている情報が多いので長くなっていますが、他のタプルはインデックス番号とI/Oアドレスの値を除いてまったく同一です(図17, 18)。このタプルの3バイト目(TPCE_INDEX: コンフィギュレーション・テーブル・インデックス・バイト)の下位6ビットにはインデックス番号が書き込まれており、このインデックス番号をCCORに書き込んだときに割り当てられるI/Oアドレスは同じタプル内のI/O空間情報構造

表3 タプル・コード一覧表

コード	タプル名称	意味
レイヤ1 Basic Compatibility Tuple		
00h	CISTPL_NULL	ヌル・タプル(無視する)
01h	CISTPL_DEVICE	デバイス情報タプル(コモン・メモリ)
02h 07h		リザーブ(デバイス情報タプルの上位互換バージョン用)
08h 0Fh		リザーブ(デバイス情報タプルの上位非互換バージョン用)
10h	CISTPL_CHECKSUM	チェックサム・コントロール用タプル
11h	CISTPL_LONGLINK_A	ロング・リンク・タプル(アトリビュート・メモリへ)
12h	CISTPL_LONGLINK_C	ロング・リンク・タプル(コモン・メモリへ)
13h	CISTPL_LINKTARGET	リンク・ターゲット・タプル
14h	CISTPL_NO_LINK	ノーリンク・タプル
15h	CISTPL_VERS_1	レベル1バージョン/製品情報タプル
16h	CISTPL_ALTSTR	各国語文字列タプル
17h	CISTPL_DEVICE_A	デバイス情報タプル(アトリビュート・メモリ)
18h	CISTPL_JEDEC_C	JEDEC デバイスIDタプル(コモン・メモリ)
19h	CISTPL_JEDEC_A	JEDEC デバイスIDタプル(アトリビュート・メモリ)
1Ah	CISTPL_CONFIG	コンフィギュレーション・タプル
1Bh	CISTPL_CFTABLE_ENTRY	コンフィギュレーション・エントリ・タプル
1Ch	CISTPL_DEVICE_OC	追加デバイス情報タプル(コモン・メモリ)
1Dh	CISTPL_DEVICE_OA	追加デバイス情報タプル(アトリビュート・メモリ)
1Eh	CISTPL_DEVICE_GEO	デバイス・ジオメトリ情報タプル
1Fh	CISTPL_DEVICE_GEO_A	デバイス・ジオメトリ情報タプル(アトリビュート・メモリ)
レイヤ2 Data Recording Format Tuple		
20h	CISTPL_MANFID	製造メーカーIDタプル
21h	CISTPL_FUNCID	機能IDタプル
22h	CISTPL_FUNCCE	機能拡張タプル
23h	CISTPL_SWIL	ソフトウェア・インタリーブ・タプル
24h 3Fh		リザーブ
40h	CISTPL_VERS_2	レベル2バージョン情報タプル
41h	CISTPL_FORMAT	フォーマット情報タプル
42h	CISTPL_GEOMETRY	ジオメトリ情報タプル
43h	CISTPL_BYTEORDER	バイト・オーダー情報タプル
44h	CISTPL_DATE	初期化日時タプル
45h	CISTPL_BATTERY	電池交換日付タプル
レイヤ3 Data Organization Tuples		
46h	CISTPL_ORG	パーティション内容情報タプル
47h 7Fh		リザーブ
レイヤ4 System-Specific Standard Tuples		
80h FEh		リザーブ
FFh	CISTPL_END	タプル・チェーン終了タプル

図13 デバイス情報タブルの例

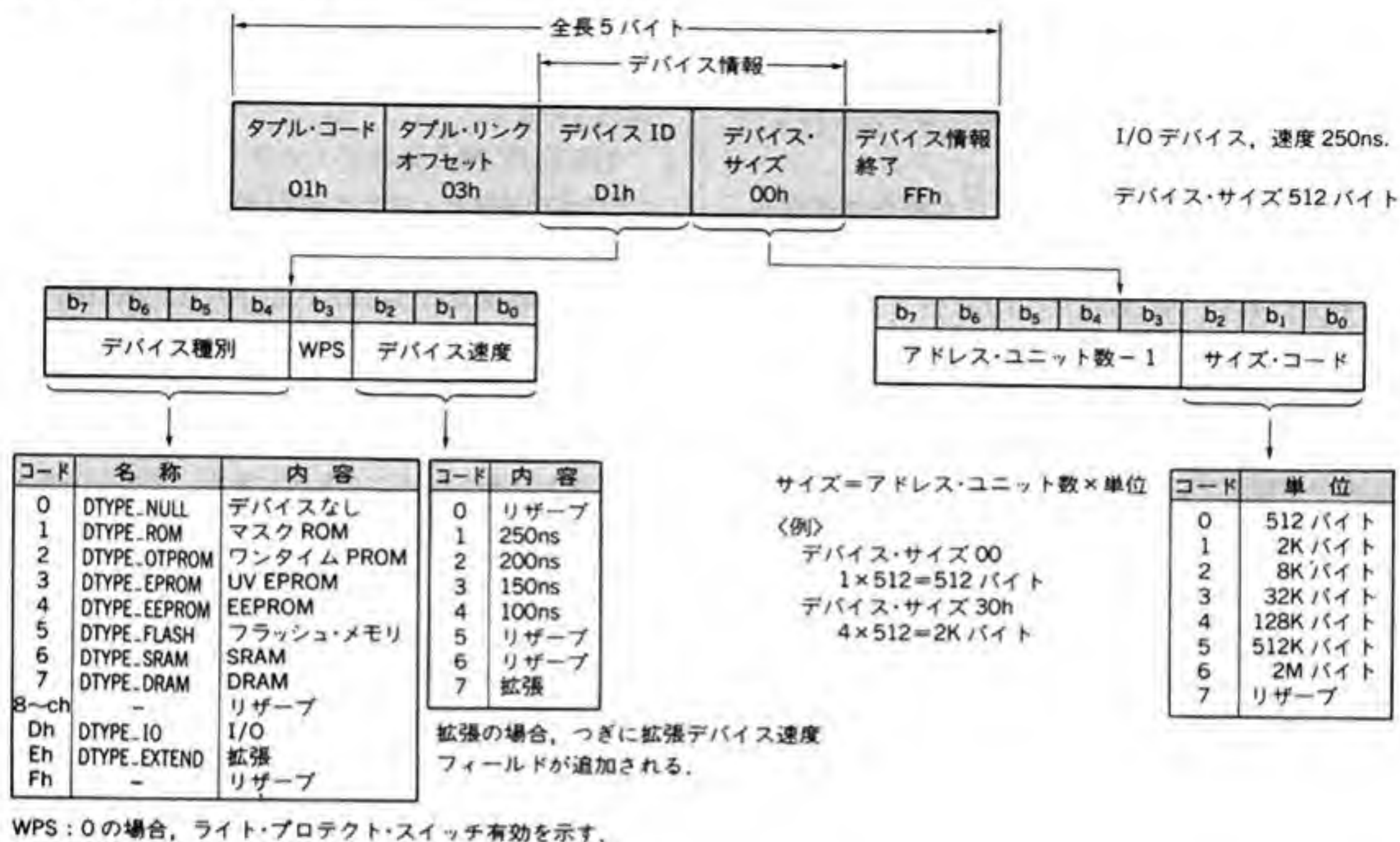


図14 レベル1バージョン/製品情報タブル

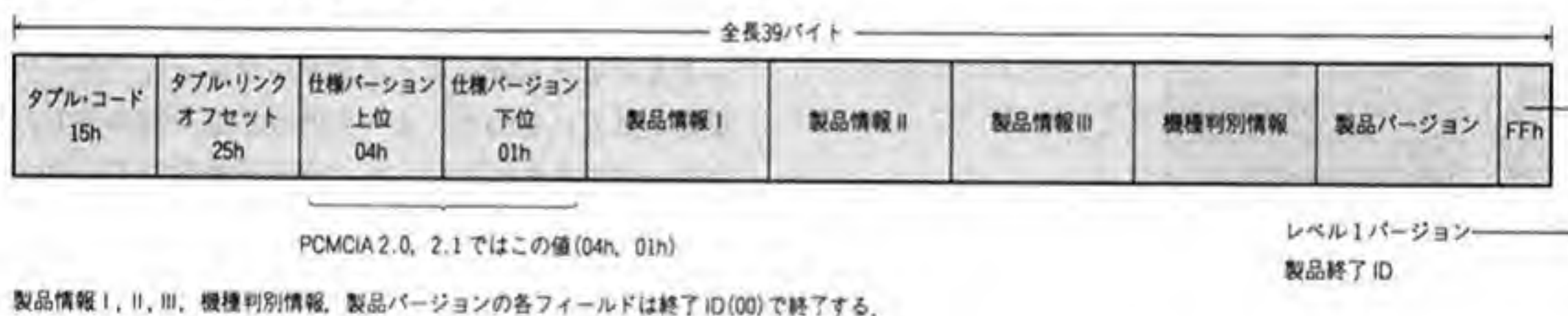
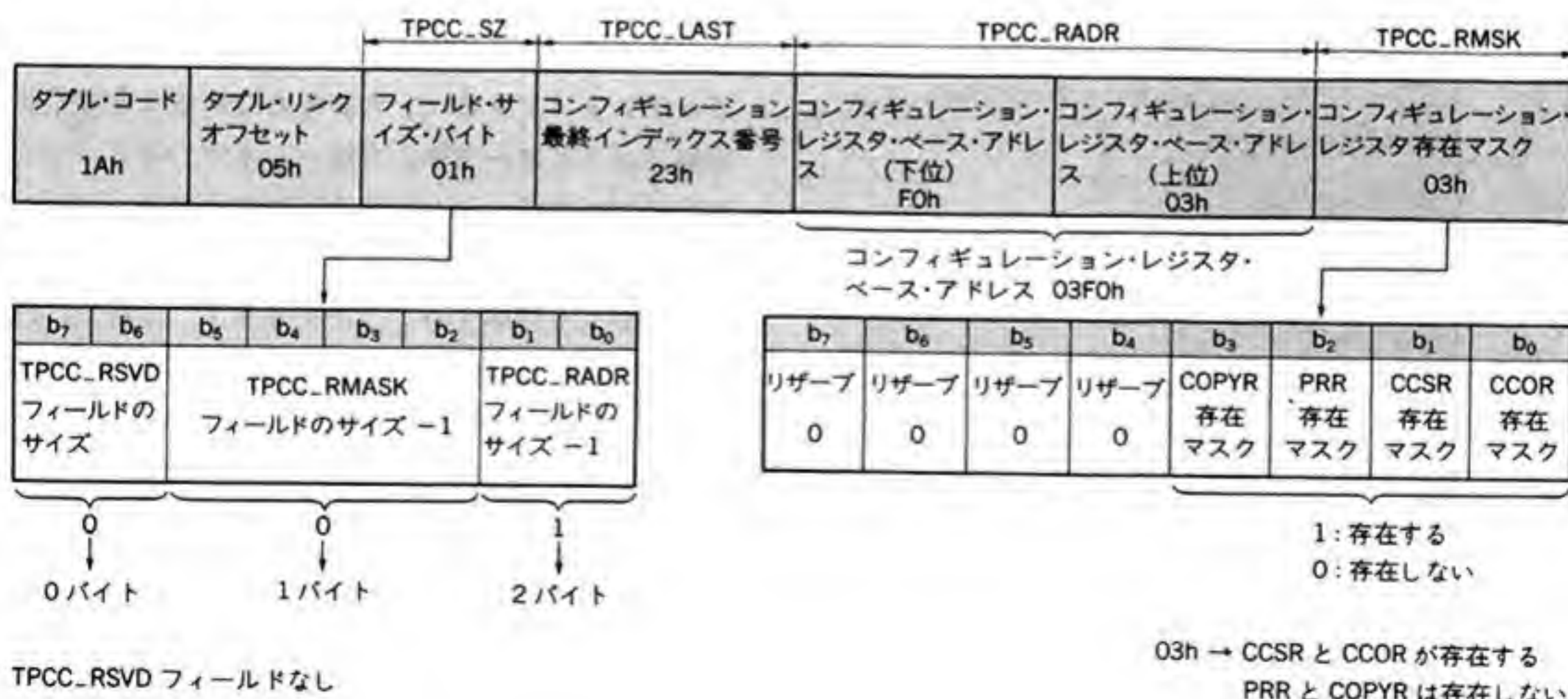


図15 コンフィギュレーション・タブルの例



体に書き込まれています。この I/O 空間情報構造体の先頭は TPCE_IO フィールドで図19のように下位 5 ビットはカード内部でデコードしているアドレス・バス信号線を LSB 側から数えた本数が書き込まれています。

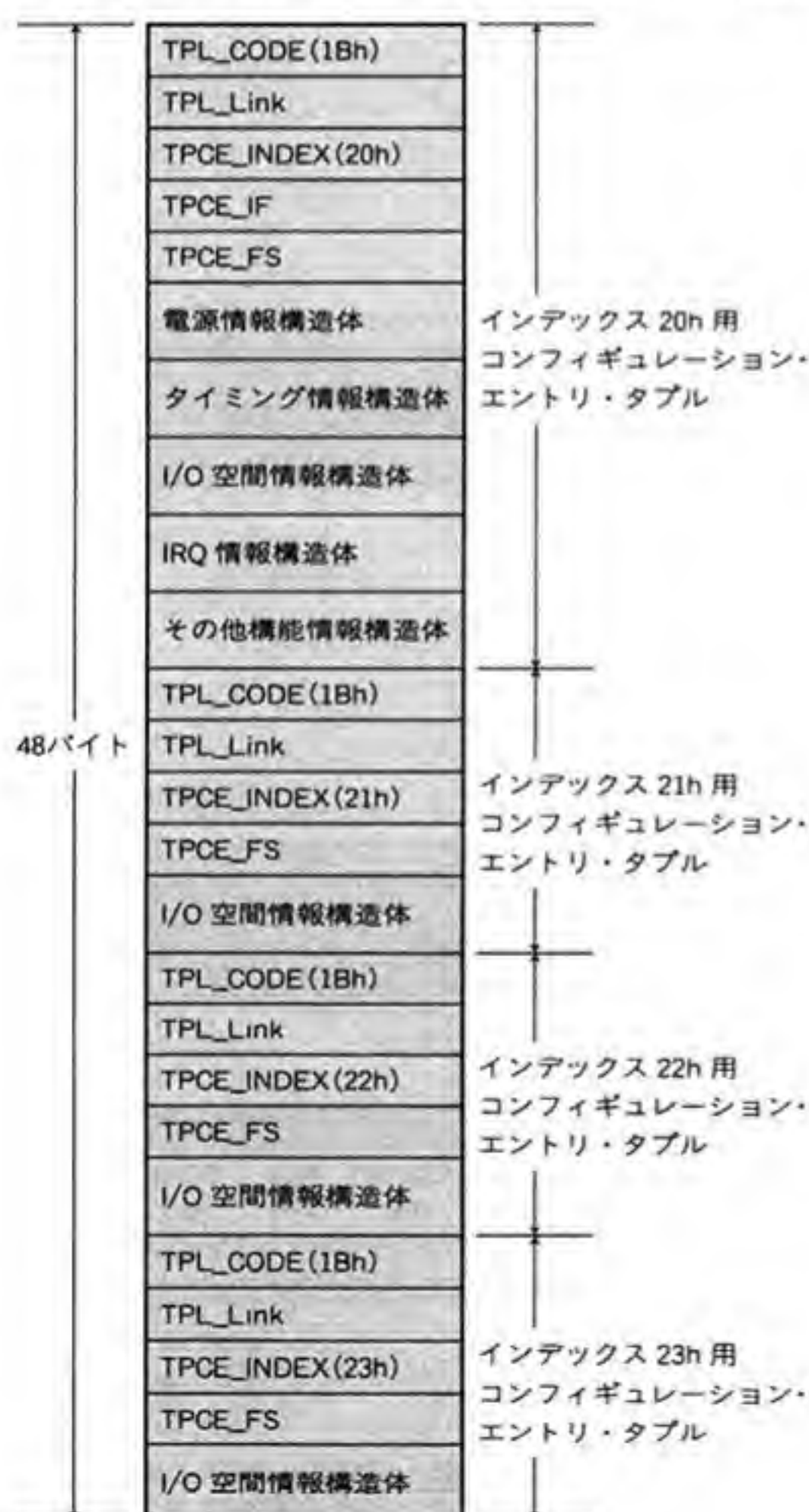
PC/AT アーキテクチャのカードの場合は通常 A₉ から A₀ の 10 本をデコードしているので“0Ah”が書き込まれています。b₈, b₅はカード内のレジスタのデータ・バス幅を示します。16 ビット幅の場合は b₈=1, b₅=0, 8 ビット幅の場合は b₈=0, b₅=1 とします。そのつぎに I/ORange ディスクリプタ・バイト、アドレス・レンジ記述ブロック(スタート・アドレスと占有バイト数を記述している)が続きます。アドレス・レンジ記述ブロックには I/O ベース・アドレスとそのアドレ

スから始まる占有バイト数が順に書き込まれています。I/O ベース・アドレスはリトルエンディアン・フォーマットで、占有バイト数は実際のバイト数-1 が書き込まれています。

TPCE_IR(割り込みディスクリプタ), TPCE_MI(その他の機能ディスクリプタ)の詳細については図20, 21 を参照してください。

CIS の最後部には CISTOL_NO_LINK(14h ノーリンク・タプル)と CISTPL_END(0FFh タプル・チェーン終了タプル)が書き込まれています。ノーリンク・タプルはアトリビュート・メモリ内に他にタプル・チェーンが存在しないことを示します。タプル・チェーン終了タプルは文字どおりタプル・チェーンの最後を示します。

図16 コンフィギュレーション・エントリ・タプル・チェーンの例



③ イネーブラ/ソケット・サービス/カード・サービス

■ イネーブラ

これまでの話は PC カード側でしたが、つぎにパソコン側から見た PCMCIA について説明します。PCMCIA 2.x では前記のように最初はスロットもカードもメモリ・カード・インターフェース・モードに設定されているので、カード内の CIS を読み出し、I/O カードであればスロット、カード両方のインターフェースを I/O モードにソフトウェアで設定しなおさなければなりません。

したがって、パソコン側ではこのソフトウェアを用意しなければなりません。このようなソフトウェアのことを「カード・イネーブラ(略してイネーブラ)」とか「コンフィギュレーション・ユーティリティ」、「クライアント・ドライバ」とか呼んでいるようですが、本稿では「イネーブラ」で統一します。イネーブラは単独のプログラムとしても常駐型プログラムとしても、またネットワーク用のデバイス・ドライバなどに組み込んだ形で作成することができます。しかし、カード内の CIS を読み出すためには DOS ベースの場合は C0000h から DFFFFh のどこかにカード内のアトリビュート・メモリをマッピングしなければなりませんから、EMS を使用している場合にはあらかじめ x オプションで使用領域がかさならないようにしておかなければなりません。また、パソコンによってはシャドウ RAM や VRAM と衝突しないように設定してお

図17 コンフィギュレーション・エントリ・タブルの例(その1)

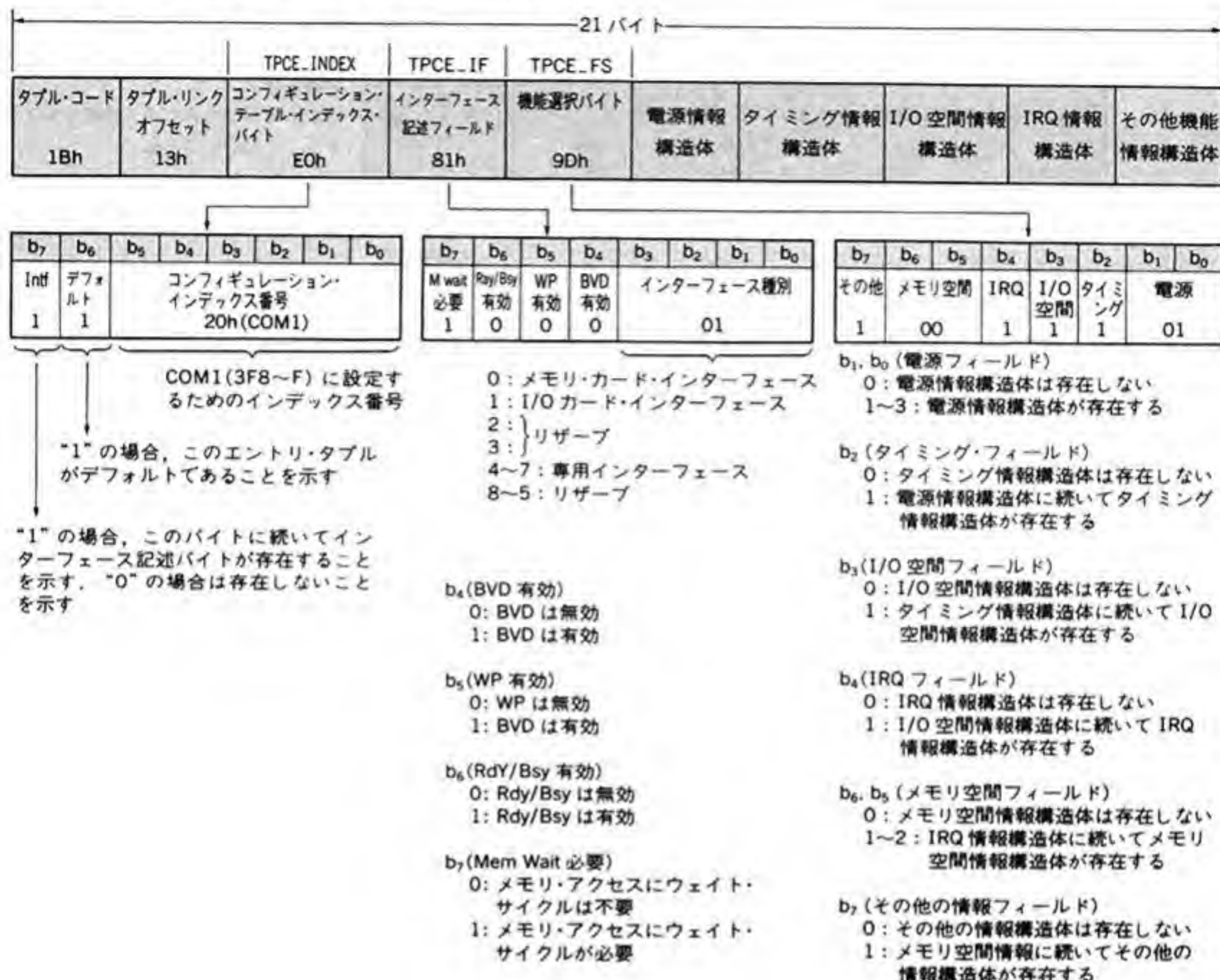


図18 コンフィギュレーション・エントリ・タブルの例(その2)

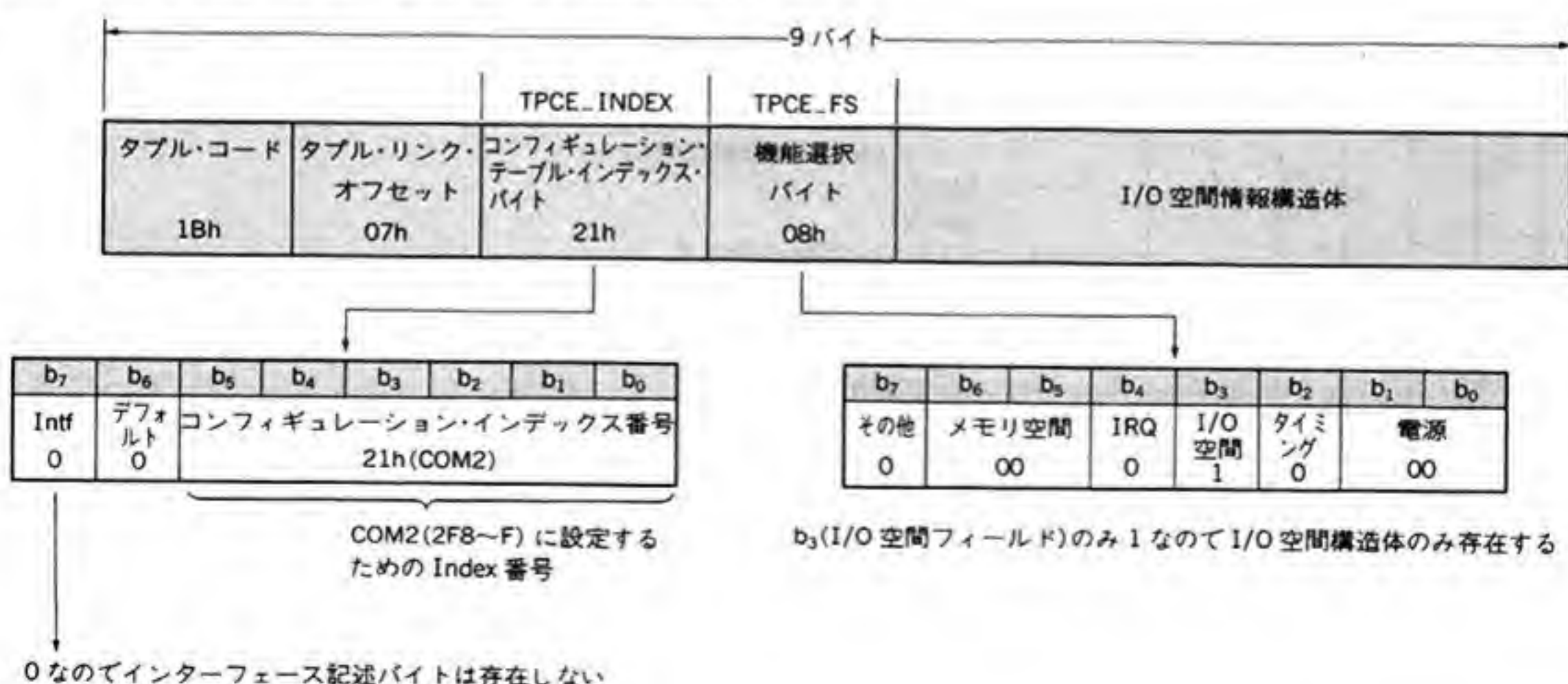


図19 I/O 空間情報構造体の例



図20 IRQ 情報構造体の例

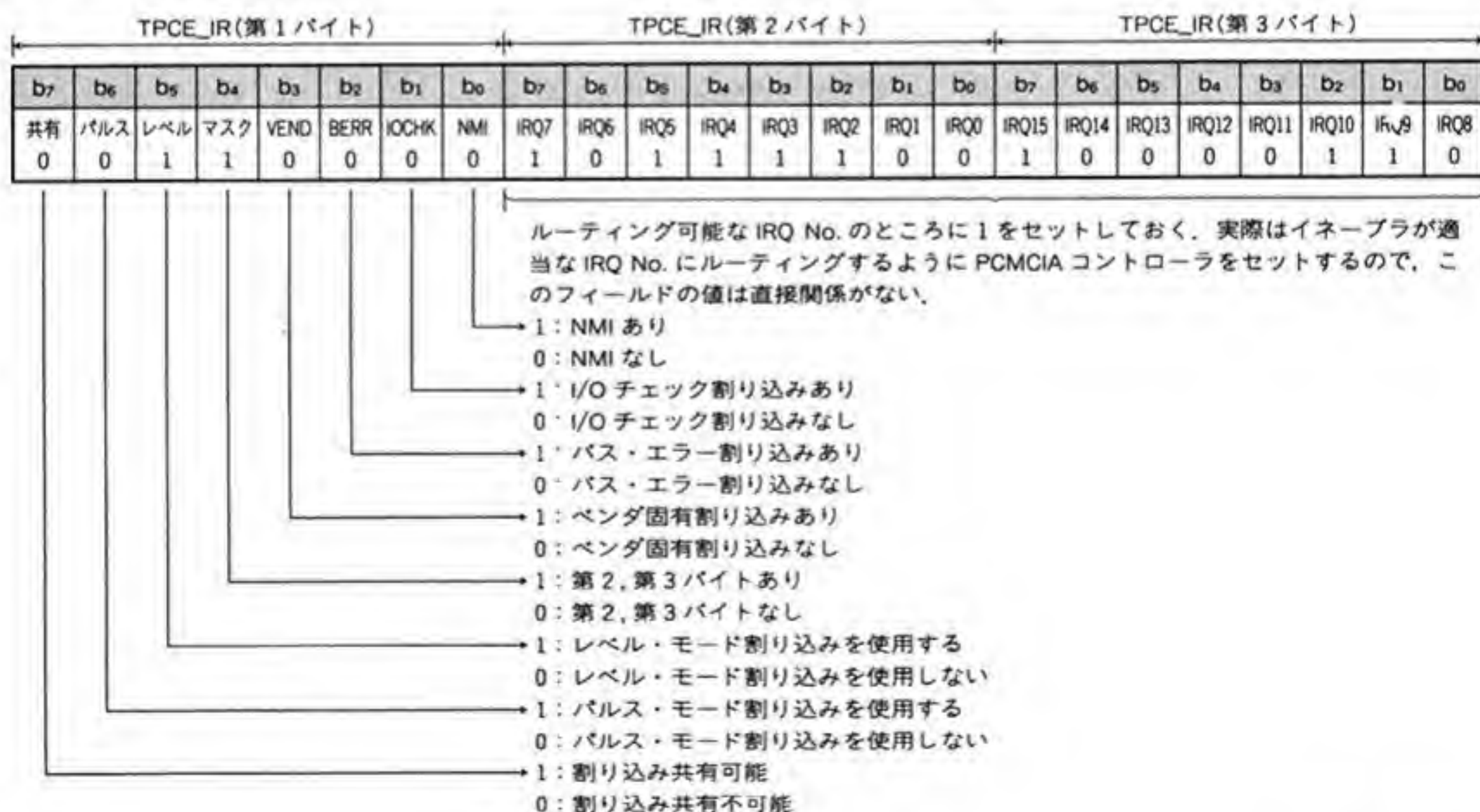


図21 その他の機能情報構造体の例

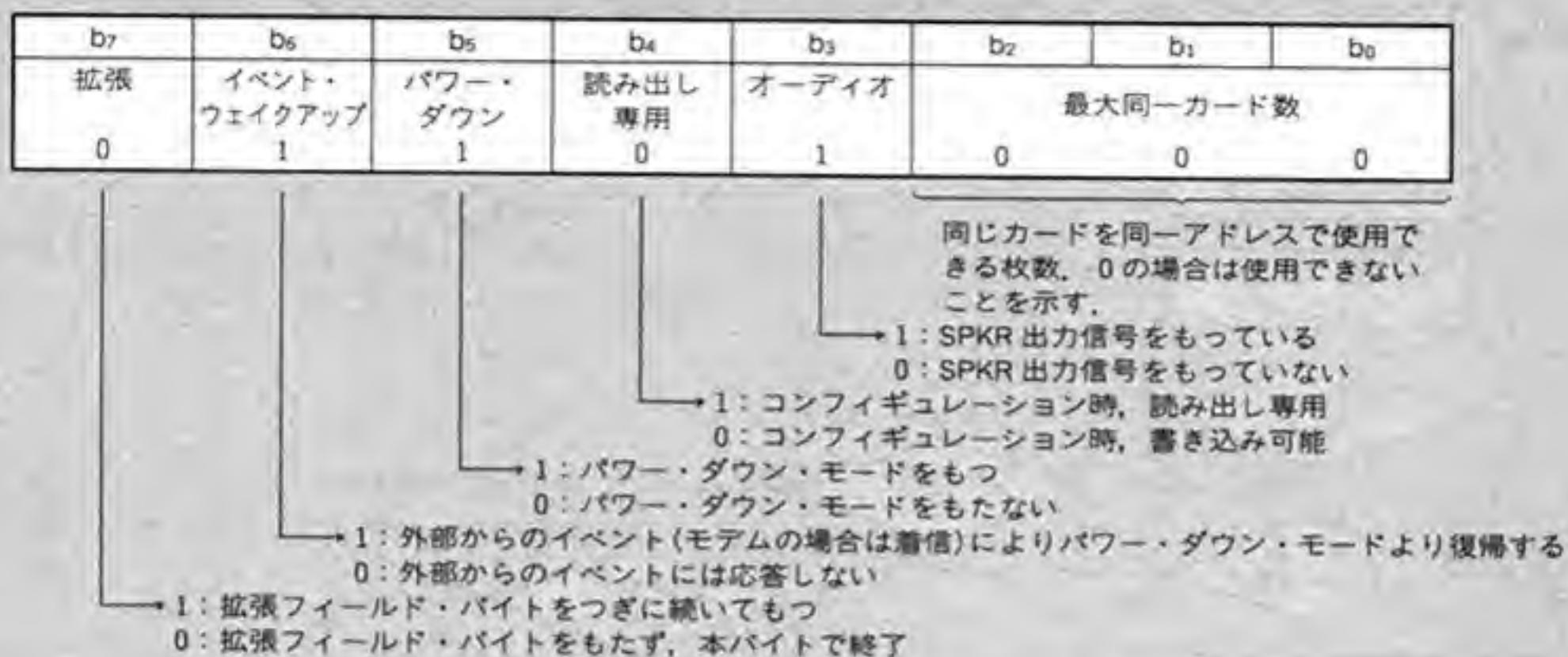


図22 モデム・カード用イネーブルのフロー
(カード・サービス、ソケット・サービスなしの場合)

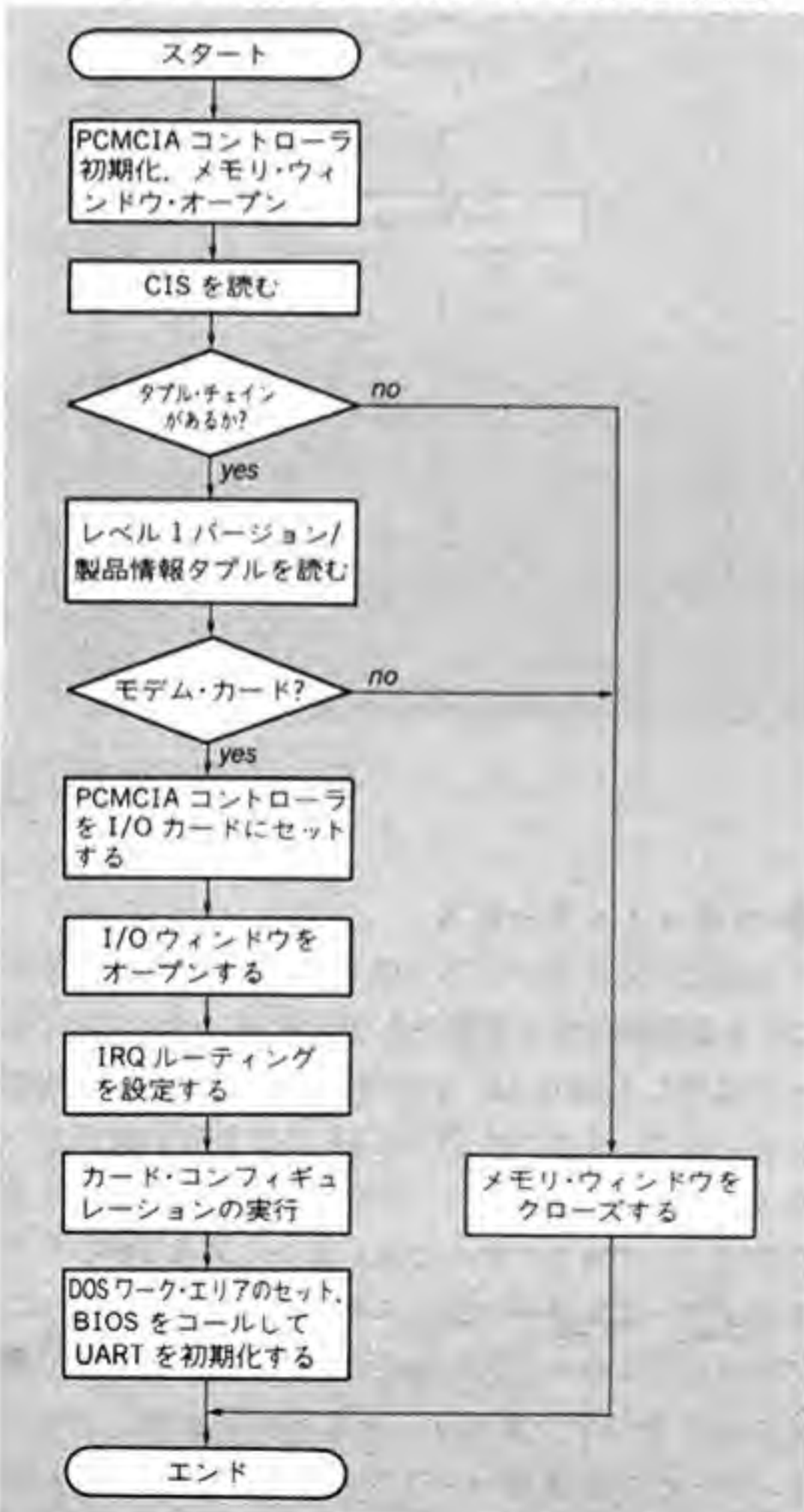


図23-1 カード・サービスを使用したモデム・カード用
イネーブラ・フロー(メイン部)

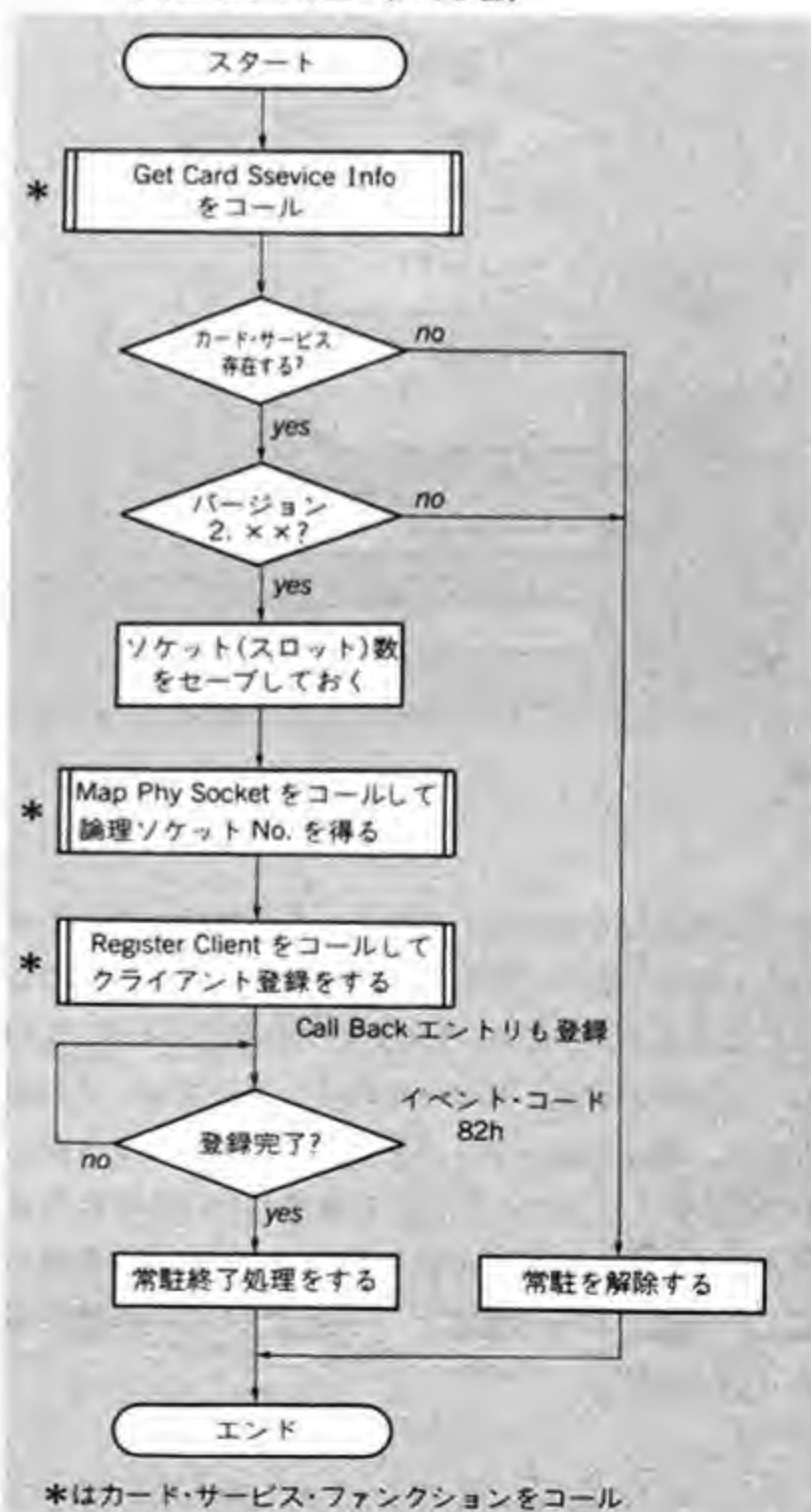
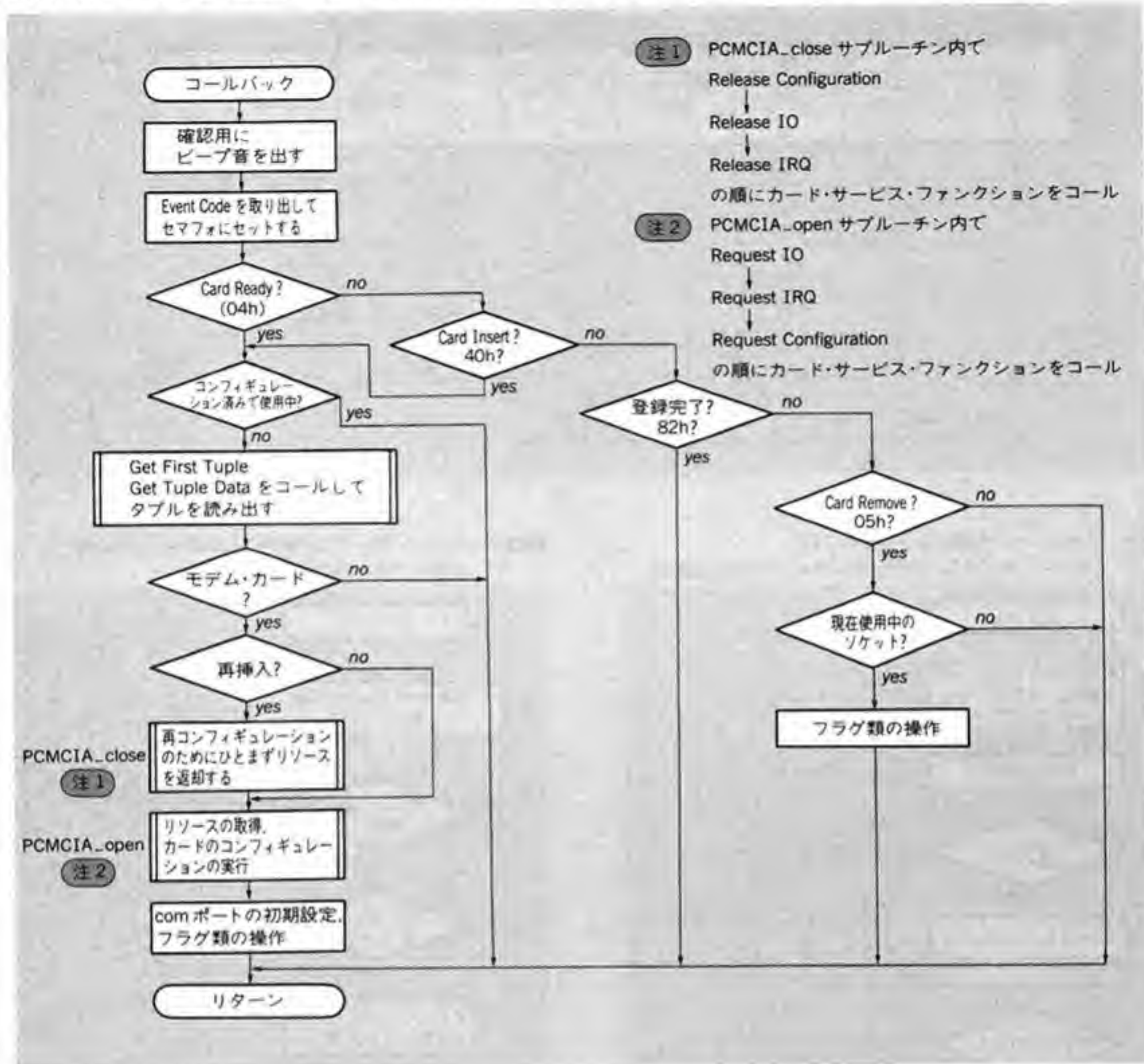


図23-2 カード・サービスを使用したモデム・カード用イネーブラ・フロー(Call Back ルーチン部)



かなければなりません。そのような下準備ができていたとしても、イネーブラのフローチャートは図22のようになります。メモリ・ウィンドウを設定したり、I/Oウィンドウの設定、スロットのインターフェースの切り替え、IRQのルーティング設定などイネーブラからPCMCIAコントローラLSIを操作しなければなりません。カードへの操作ではメモリ・ウィンドウを通してCISを読み出して解釈したうえでCCRへの書き込みを行います。

■ ソケット・サービス

このようにイネーブラではPCMCIAコントローラLSIを直接操作する必要がありますが、パソコンによって必ずしも同じLSIを使用していないという問題があります。そのためPCMCIAではLSIが違ってしまうようなパソコン側のハードウェアの違いを吸収するために「ソケット・サービス」というドライバ・ソフトウェア・インターフェースを規格化しています。このソケット・サービスも現在のバージョンは2.1で表4のようなファンクションをもっています。ソケット・サービスはDOSベースではデバイス・ドライバと

して登録されメモリ上に常駐します。呼び出し方法は[AH]に表4のファンクション・コードを、他のレジスタにパラメータを入れたのち「INT 1Ah」で呼び出して実行させます。イネーブラからソケット・サービスを呼び出せば PCMCIA コントローラ LSI に直接アクセスする必要がなくなるので、パソコンの機種に依存しない互換性の高いイネーブラを作成することができます(図23 参照)。

■ カード・サービス

このようにソケット・サービスにはいろいろと便利などころもあるのですが、つぎのような欠点もあります。第一に CIS を読み出して解釈する場合、アトリビュート・メモリに直接イネーブラがアクセスしなければならないという点です。つまり CIS の読み出しをもっと簡単に(たとえばダブル・コードを指定するだけでダブルを読み出してくれるような)したいということです。第二にソケット・サービスではリソース(メモリ・ウィンドウ、I/O ウィンドウなど)の管理をまったく行っていないという点です。たとえば、スロットが2個あるパソコン上で片方のスロットで LAN カードがメモリ・ウィンドウ C8000 から CFFFF まで、I/O ウィンドウを 300h から 16 バイト使用中にもう一方のスロットのカードのためにメモリ・ウィンドウを C8000h から C8FFFh まで取得しようとしてもエラーとはならないという問題があります(I/O ウィンドウや IRQ に関しても同様)。したがって、いろいろなドライバが常駐し、いろいろなカードを差し替えて使用するようなマルチクライアント環境では同じメモリ・ウィンドウに両方のカードのメモリが現れたり、同じ I/O アドレスに両方のカードのレジスタが現れたり、カードから IRQ が発生したときにどちらのドライバの割り込み処理ルーチンにジャンプするのかがわからなくなり、ドライバが暴走してしまうようなことになります。

PCMCIA ではこのような問題に対してソケット・サービスを拡張するのではなく、あらたに「カード・サービス」というソフトウェア・インターフェースを規格化することで対応しています。カード・サービスは図23のようにイネーブラとソケット・サービスの間に位置し、イネーブラからは直接ソケット・サービスをコールせずにカード・サービスをコールします。カード・サービスはイネーブラから要求されたファンクシ

表4 ソケット・サービス・ファンクション一覧表

名称	コード	内 容
GET_ADP_CNT	80h	ソケット・サービスでサポートしているソケット数を取り出す
	81h	リザーブ
	82h	リザーブ
GET_SS_INFO	83h	ソケット・サービスの情報(バージョンなど)を取り出す
INQ_ADAPTER	84h	アダプタの仕様、能力を取り出す
GET_ADAPTER	85h	指定されたアダプタの現在の設定内容を取り出す
SET_ADAPTER	86h	指定されたアダプタを設定する
INQ_WINDOW	87h	指定されたウィンドウの仕様、能力を取り出す
GET_WINDOW	88h	指定されたウィンドウの設定内容を取り出す
SET_WINDOW	89h	ウィンドウ(メモリ、I/O)の設定を行う
GET_PAGE	8Ah	メモリ・ウィンドウ内に割り当てられているページの現在の設定内容を取り出す
SET_PAGE	8Bh	メモリ・ウィンドウ内にカード内のメモリを割り当てる
INQ_SOCKET	8Ch	指定されたソケットの仕様、能力を取り出す
GET_SOCKET	8Dh	指定されたソケットの現在の設定内容を取り出す
SET_SOCKET	8Eh	指定されたソケットの設定を行う
GET_STATUS	8Fh	指定されたソケット、カードのステータスを取り出す
RESET_SOCKET	90h	指定されたソケット内のカードをリセットして電源 ON 直後の状態に戻す
	91h	リザーブ
	92h	リザーブ
	93h	リザーブ
INQ_EDC	94h	リザーブ
	95h	EDC(メモリのエラー修復コード)ジェネレータの仕様、能力を取り出す
GET_EDC	96h	EDC ジェネレータの設定内容を取り出す
SET_EDC	97h	EDC ジェネレータを設定する
START_EDC	98h	EDC ジェネレータをスタートさせる
PAUSE_EDC	99h	EDC ジェネレータをいったん停止させる。
RESUME_EDC	9Ah	いったん停止している EDC ジェネレータを元に戻す
STOP_EDC	9Bh	EDC ジェネレータを停止させる
READ_EDC	9Ch	EDC ジェネレータの計算結果を取り出す
GET_VENDOR_INFO	9Dh	ソケット・サービスのベンダ固有の情報を取り出す
ACK_INTERRUPT	9Eh	ソケットの状態の変化に関する情報を取り出す
PRIOR_HANDLER	9Fh	アダプタに対して優先権をもつハンドラのエントリ・ポイントを取り出す
SS_ADDR	0A0h	ソケット・サービス自身が使用しているコード領域、データ領域のディスクリプタを取り出す
ACCESS_OFFSETS	0A1h	I/O ポートを通してカード内のメモリにアクセスできるように設定する
	0A2h	リザーブ
	0A3h	リザーブ
	0A4h	リザーブ
	0A5h	リザーブ
	0A6h	リザーブ
	0A7h	リザーブ
	0A8h	リザーブ
	0A9h	リザーブ
VEND_SPECIFIC	0AEh	ベンダ独自ファンクション

PCMCIA のカード・サービスの規格では図24 のようにカード・サービスに対してリクエストを出すプログラムのことをクライアントと呼んでいます。前記の

[illegible]

このほか、登録済みのクライアントに対してカード挿入/抜去の検出、カードのバッテリ電圧降下などのイベントが発生するとクライアントをコールバックして通知します。

たとえば、スロットが一つだけのパソコンで Windows 3.1 を動かしているとします。LAN カードをスロットに入れ、TCP/IP プロトコルを使用して FTP でファイル転送を実行した後、カードを FAX/モデム・カードに差し替えて通信ソフトウェアを実行する。さらにその後、再び LAN カードに差し替えて FTP を使用する。このようなことが Windows 3.1 を終了させずに行えることを「活線挿抜ができる」といいます。このことを実現するためにはカード挿入/排出を検出し、挿入時に自動的にコンフィギュレーションを実行するモデム・カード用のイネーブラと LAN カード用のドライバが常駐していなければなりません。また、PC カードは抜くだけではなく電源を落とした場合、再度電源を投入してもメモリ・カードに戻ったままなのでノート・パソコンのレジューム機能にも対応しなければなりません。つまり、レジュームから復帰した時点で I/O カードに自動的にコンフィギュレーションされなければなりません。活線挿抜にきちんと対応

表5 カード・サービス・ファンクション一覧

① クライアント・サービスに関するファンクション

ファンクション名	コード	内 容
GetCardServicesInfo	0Bh	カード・サービスの情報を取り出す
RegisterClient	10h	クライアントの登録を行う
DeregisterClient	02h	クライアントの登録を解除する
GetStatus	0Ch	カードとソケットの現在のステータスを取得
ResetCard	11h	指定のソケット内のカードをリセットする
SetEventMask	31h	クライアントに通知するイベントのマスク設定
GetEventMask	2Eh	クライアントに通知するイベントのマスク解除

② リソース管理に関するファンクション

ファンクション名	コード	内 容
RequestIO	1Fh	I/O リソースの取得
ReleaseIO	1Bh	I/O リソースの返却
RequestIRQ	20h	IRQ リソースの取得
ReleaseIRQ	1Ch	IRQ リソースの返却
RequestWindow	21h	システム・メモリ内のメモリ・ウィンドウの取得
ReleaseWindow	1Dh	システム・メモリ内のメモリ・ウィンドウの返却
ModifyWindow	17h	取得済みのメモリ・ウィンドウの属性を変更する
MapMemPage	14h	カード内のメモリをメモリ・ウィンドウ内に割り当てる
RequestSocketMask	22h	ステータス・チェンジのマスクを設定する
ReleaseSocketMask	2Fh	ステータス・チェンジのマスクを解除する
RequestConfiguration	30h	カード・コンフィギュレーションの実行
GetConfigurationInfo	04h	コンフィギュレーション情報を得る
ModifyConfiguration	27h	コンフィギュレーションを変更する
ReleaseConfiguration	1Eh	カード・コンフィギュレーションを解除する

③ メモリ・カード内のメモリ・ブロックに関するファンクション

ファンクション名	コード	内 容
OpenMemory	18h	メモリ・ブロックをオープンする
ReadMemory	19h	オープン済みのメモリ・ブロックから読み出す
WriteMemory	24h	オープン済みのメモリ・ブロックに書き込む
CopyMemory	01h	オープン済みのメモリ・ブロック間でコピーをする
RegisterEraseQueue	0Fh	メモリ・ブロック消去のためのキューを登録する
CheckEraseQueue	26h	メモリ・ブロック消去のためのキューを確認する
DeregisterEraseQueue	25h	メモリ・ブロック消去のためのキューを解除する
CloseMemory	00h	メモリ・ブロックをクローズする

④ クライアント用ユーティリティ・ファンクション

ファンクション名	コード	内 容
GetFirstTuple	07h	指定コードのタプルの最初の位置を得る
GetNextTuple	0Ah	指定コードのタプルのつぎのタプルの位置を得る
GetTupleData	0Dh	指定コードのタプルを読み出す
GetFirstRegion	06h	メモリ・ブロックの最初の位置を得る
GetNextRegion	09h	つぎのメモリ・ブロックの位置を得る
GetFirstPartition	05h	パーティション・ブロックの最初の位置を得る
GetNextPartition	08h	つぎのパーティション・ブロックの位置を得る

⑤ クライアント・サービスに関する拡張ファンクション

ファンクション名	コード	内 容
ReturnSSEntry	23h	ソケット・サービスのエントリ・ポイントを取り出す
MapLogSocket	12h	論理ソケット番号を物理ソケット番号に変換
MapPhySocket	15h	物理ソケット番号を論理ソケット番号に変換
MapLogWindow	13h	ウィンドウ・ハンドルを物理ウィンドウに変換
MapPhyWindow	16h	物理ウィンドウのウィンドウ・ハンドルに変換
RegisterMTD	1Ah	MTD(メモリ・ドライバ)を登録する
RegisterTimer	28h	一定時間ごとにコールバックされるよう登録する
SetRegion	29h	CIS 内に存在しないメモリ・ブロックを設定
ValidateCIS	2Bh	CIS の検証を行う
RequestExclusive	2Ch	あるクライアントに特別使用権を与える
ReleaseExclusive	2Dh	あるクライアントの特別使用権を解除する
GetFirstClient	0Eh	最初のクライアント・ハンドルを取り出す
GetNextClient	2Ah	つぎのクライアント・ハンドルを取り出す
GetClientinfo	03h	クライアントに関する情報を取り出す
AddSocketServices	32h	新たなソケット・サービスを追加する
ReplaceSocketServices	33h	現在使用中のソケット・サービスを置換する
VendorSpecific	34h	ベンダ独自ファンクション
AdjustResourceInfo	35h	リソース・テーブルを整理する

していればレジュームからの復帰時にカード・サービスからコールバックされるので問題はありません。

カード・サービスは、このようにリソース管理、イベント管理という二つの面で重要な役割をはたしています。

④ PC カードの現状の問題点と今後の方向

現時点ではノート・パソコン、サブノート・パソコンについてカタログ上では PCMCIA 準拠のスロットをもっているということは明記されています。しかし、なかには Type だけを表示し PCMCIA の規格のバージョンを表示していないものがあるので、カタログからだけでは I/O カードが使えるのかどうか不明な場合があります。

また、ソケット・サービスやカード・サービスが付属しているのかどうかという点については、カタログ上ではまったく触れられていないため購入してみなければわからないというのが現状です。また、現実にはソフトウェア・ライセンスなどの関係で古いバージョンの規格にしか対応していないカード・サービスやソケット・サービスあるいは独自規格のものをバンドルしているノート・パソコンもあって、よりいっそう混乱を大きくしています。IBM からの PC DOS 6.1 J/V には Phoenix 製のソケット・サービス、カード・サービスがバンドルされるようになりましただので、カード・サービスやソケット・サービスが付属していなかったマシンでも使用できるようになりました。しかし、PCMCIA コントローラ LSI がインテル製でないと動作しないという問題も残っています。どのノート・パソコンも PCMCIA 2.1 準拠のカード・サービスやソケ

ット・サービスをバンドルしてくれるようになれば「Plug & Play」に近づくのですが、現状はまだ1、2年先という感じです。

パソコン側がはっきりしないために PC カードのベンダ側もいろいろなバージョンのカード・サービス(バージョンが同じでもカード・サービスのベンダにより少し異なる部分もある)に対応させたイネーブラやドライバを供給せねばならず、個々の機種での動作確認に大変手間取っています。現時点ではカード・サービスを使用せず、インテル 82365 を直接制御しているイネーブラやドライバが8割以上ですが、パソコン側が PCMCIA 2.1 のカード・サービスに統一されてくれば PC カード側もそれに合わせて統一されてくるものと思われま

す。PCMCIA では PC カード・タイプのハード・ディスクやフラッシュ・メモリ・カードからブートするための規格や PC カードの中にデバイス・ドライバを入れておいて、カード挿入時そこからロードするための規格(実行形式の ROM イメージは XIP 規格)などが提案されています。「Plug & Play」を実現するためにはいままで以上にソフトウェアが大変になります。そのために早く PCMCIA 2.1 準拠のカード・サービスとそれに対応した PC カードのイネーブラやドライバが広まることを期待します。

参考文献

- 1) 「PCMCIA 2.1 規格書」、1993 年 7 月、PCMCIA 事務局発行
- 2) 「IC メモリカードガイドライン Ver 4.1 仕様書」、平成 3 年 9 月、(社)日本電子工業振興協会発行
- 3) 「MBH10213 モデムカード仕様書」、1993 年 10 月、富士通(株)発行

おかむら・ちかよし ラトックシステム㈱

- 本書掲載記事の利用についてのご注意 — 本書掲載記事には著作権があり、また工業所有権が確立されている場合があります。したがって、個人で利用される場合以外は所有者の承諾が必要です。
また、掲載された回路、技術、プログラムを利用して生じたトラブル等については、小社ならびに著作権者は責任を負いかねますのでご了承ください。
- 本書に関するご質問について — 文章、数式等の記述上で不明な点についてのご質問は、必ず往復はがきか返信用封筒を同封した封書にてお願いいたします。ご質問は著者に回送し直接回答していただきますので、多少時間がかかります。また、本書の範囲を超えるご質問には応じられませんので、ご了承ください。

OPENDesign No.1

1994年3月1日 初版発行

1996年6月20日 第4版発行

編集人 金子俊夫

発行人 蒲生良治

発行所 CQ 出版株式会社

〒170 東京都豊島区巣鴨 1-14-2 CQビル

電話 編集部 03-5395-2147 販売部 03-5395-2141

広告部 03-5395-2133

振替 00100-7-10665

写植・版下 有みやこワードシステム

印刷・製本 東京書籍印刷株

©1994 CQ Publishing Co. Ltd.

Printed in Japan

本誌からの無断転載を禁じます。

落丁・乱丁の場合はお取り替えいたします。
定価は表4に表示してあります。

オープン・システム実践教室

事例に学ぶマルチベンダ・システム構築法

小暮 裕明 編著 B5判 152頁 2色刷(一部) 定価1,650円

最近、オープン・システムとかダウンサイジングといった言葉が頻繁に登場しています。しかし、言葉がよく聞かれるわりには、マルチベンダ・システムの開発に関わる実践技術の情報となるとほとんどが断片的なものであり、体系的に整理されたものはなかなか手にすることが出来ません。そこで、本書では、汎用機からパソコンまで、広い意味でのシステムエンジニアを対象に、オープン・システム開発のための技術を、多くのケーススタディを交えながら、開発の現場で役立つように体系づけて解説します。オープン・システムの構築では、なにを・どう選ぶかという、「製品選択」も重要な鍵です。そこで、本書では、製品情報もていねいにフォローします。

＜内容＞ オープン・システム構築の現場から／まず手づくりLANからスタート／ホストからの発想タイプ☆全国販売管理システム／PC-LANタイプ☆本格的社内OAシステム／LANビジネス展開タイプ☆サーバ／重化装置／短期決戦タイプ☆報道情報システム／シームレスからボーダレスへ☆キャンパス・ネットワーク／オープン分散システムへのステップ／ネットワーク構築事例集／座談会☆オープン・システム



オープン・システム入門教室

ダウンサイジングに対処するための10科目

小暮裕明 著 B5判 2色刷 176頁 定価1,650円

本書では、広い範囲のオープン・システム技術を、ドキュメント書法やSQL、開発言語は「国語」に、コンピュータ史やUnixの勢力図、分散/集中の歴史は「社会」に、信頼性計算やトランザクション量は「数学」に、マルチメディアは「音楽・美術」に……というぐあいに、学生時代におなじみだった、国・算・理・社・英・音・家・技・体・倫社の10科目に分割、システム・インテグレータをめざす技術者のために、オープン・システム技術のポイントを整理します。

SEスキルアップNOTE

上級ソフトウェア技術者になるための実践計画

小暮裕明 著

B5判 160頁

定価1,650円

本書では、10年計画で上級SEになる、という目標を設定し、「プログラミング技術」からはじまって、「ドキュメント技術」、「ソフトウェア工学」、「折衝技術」、「見積り・プロジェクト管理」、「企画/開発力」といった順序で、SEがマスターしておくべき技術を、ケース・スタディをまじえながら、実践的に紹介していきます。

読者対象は、これからSEをめざそうというフレッシュマン、および新人を教育しなければならない立場にあるベテラン技術者/管理職の方々です。

マイコン技術者 スキルアップ事典

ハードに強いエンジニアになるためのデータバンク

長嶋洋一 著

B5判 2色刷(一部) 180頁

定価1,650円

本書は、「技術不安」を解消して、自信をもってエンジニア人生に出帆してほしい、という願いから企画されました。全体の構成は、＜イントロダクション＞で「マイコン技術者の仕事地図」を示し、ついで個々の技術内容を＜事典編＞で整理しました。事典は、＜基礎＞、＜パソコン活用＞、＜マイコン開発＞、＜チップ関連＞、＜信号と信頼性＞、＜情報収集とドキュメント技術＞、＜ASIC＞、＜分散処理＞の8項目です。



ベースはSCSI-2. SCSIの動作原理
おもな周辺装置の動作とSCSIコマンド
パソコンやWSの機種別SCSI研究,そして
デバッグ・ツールやトラブル対策について詳しく解説
SCSI-3最新情報やPCMCIAの詳細解説も収録

CQ出版社 **オープンデザイン No.1** 定価1,800円(本体1,748円)

ISBN4-7898-3524-3 C3055 P1800E